OPERATIONS RESEARCH
Vol. 53, No. 2, March–April 2005, pp. 197–218
ISSN 0030-364X | EISSN 1526-5463 | 05 | 5302 | 0197

DOI 10.1287/opre.1040.0170

Maximum Pressure Policies in Stochastic Processing Networks

J. G. Dai, Wuqin Lin
School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, Georgia 30332-0205
{dai@isye.gatech.edu, linwq@isye.gatech.edu}

Complex systems like semiconductor wafer fabrication facilities (fabs), networks of data switches, and large-scale call centers all demand efficient resource allocation. Deterministic models like linear programs (LP) have been used for capacity planning at both the design and expansion stages of such systems. LP-based planning is critical in setting a medium range or long-term goal for many systems, but it does not translate into a day-to-day operational policy that must deal with discreteness of jobs and the randomness of the processing environment.

A stochastic processing network, advanced by J. Michael Harrison (2000, 2002, 2003), is a system that takes inputs of materials of various kinds and uses various processing resources to produce outputs of materials of various kinds. Such a network provides a powerful abstraction of a wide range of real-world systems. It provides high-fidelity stochastic models in diverse economic sectors including manufacturing, service, and information technology.

We propose a family of maximum pressure service policies for dynamically allocating service capacities in a stochastic processing network. Under a mild assumption on network structure, we prove that a network operating under a maximum pressure policy achieves maximum throughput predicted by LPs. These policies are semilocal in the sense that each server makes its decision based on the buffer content in its serviceable buffers and their immediately downstream buffers. In particular, their implementation does not use arrival rate information, which is difficult to collect in many applications. We also identify a class of networks for which the nonpreemptive, non-processor-splitting version of a maximum pressure policy is still throughput optimal. Applications to queueing networks with alternate routes and networks of data switches are presented.

Subject classifications: production/scheduling: sequencing, stochastic; queues: networks.
Area of review: Stochastic Models.
History: Received May 2003; revision received January 2004; accepted March 2004.

1. Introduction

In a series of three papers, J. Michael Harrison (2000, 2002, 2003) introduced progressively more general stochastic models, called *stochastic processing networks*. These networks are much more general than multiclass queueing networks that have been the subject of intensive study in the research community in the last 15 years. See, for example, Bertsimas et al. (1994), Bramson (1998), Chen and Yao (1993), Dai (1995), Harrison (1988), Harrison and Nguyen (1993), Kumar and Kumar (1994), Meyn (1997), Wein (1992), and Williams (1998).

Loosely speaking, an open processing network is a system that takes inputs of materials of various kinds and uses various processing resources to produce outputs of materials of various (possibly different) kinds. Here, material is used as a generic substitute for a variety of entities that a system might process, such as jobs, customers, packets, commodities, etc.; we use material and job interchangeably in the rest of this paper. In fact, material need not be discrete, although we will focus on the case when all jobs are discrete. Typically, there are constraints on the amount of material that a given server can process in a given time period. In addition, material may be processed by several servers, may be split up, or combined with other kinds of materials before a final output is produced. Control is exerted through allocations of processors for the processing of one or more kinds of material.

As observed in Bramson and Williams (2003), deterministic (or average) models for describing such processing networks have a long history in economics and operations research. For example, the book *Activity Analysis of Production and Allocation*, edited by T. C. Koopmans (1951), provides an excellent summary of the early stages of development of such models. A prominent role is played there by the notion of a *processing activity*, which consumes certain kinds of material, produces certain (possibly different) kinds of materials, and uses certain processors in the process. In a sense, Harrison's (2000) model of an open stochastic processing network is a stochastic analog of dynamic deterministic production models such as those first considered by Dantzig (1951).

In this paper, we focus on a special class of Harrison's model. Even this specialized class of stochastic processing network models is broad enough to cover a wide range of application fields, including manufacturing systems, service systems, computer systems, and computer communication

networks. In addition to many systems that can be modeled by multiclass queueing networks, the added features of a stochastic processing network can model many new elements like machine-operator interaction and material handling in a semiconductor wafer fabrication facility, cross-trained workers at a call center, networks of data switches, parallel processing computer systems, and routing in the Internet. Section 9 provides more detailed descriptions for some applications, including networks of data switches and queueing networks with alternate routes. Readers are encouraged to jump to this section to get a feel of the scope of the application domains.

We are interested in the dynamic control of these stochastic processing networks at the operational level so that their long-term objectives are met. Two types of long-term objectives are often considered in the literature: (1) maximizing system throughput, and (2) minimizing work in process or delay of the system. In this paper, we will focus on the first objective.

The maximum throughput or processing capacity of a system is often constrained by the processing speed of bottleneck resources. These constraints can be turned into a linear program (LP) from which the system processing capacity can be determined. This approach has been used in practice in capacity planning at either the design or expansion stage of a system. See, for example, Thomas and McClain (1993). LPs based planning is very much relevant in setting a medium range or long-term goal for the system. Because servers have overlapping processing capabilities, sometimes it may not be possible for any operational policy to achieve the maximum throughput predicted by the LP (see the example in §7). Indeed, even in the multiclass queueing network setting, it is now well known that many commonly used service policies including first in first out (FIFO) are not throughput optimal (see Bramson 1994 and Seidman 1994).

In this paper, we propose a family of operational policies called *maximum pressure policies*. There are two versions of these policies, depending on whether processor splitting is allowed in the policy space. We prove that both versions of these policies are throughput optimal under an extreme-allocation-available (EAA) assumption on the network structure. The assumption is satisfied for a wide class of networks. Such networks include multiclass queueing networks, parallel server systems, networks of data switches, and queueing networks with alternate routes. In addition, we explicitly characterize, through LPs, the stability regions of stochastic processing networks operating under a maximum pressure policy.

Our maximum pressure policies are generalizations of some forms of MaxWeight policies studied in Andrews et al. (2004) and Stolyar (2004) in the network setting. In their papers, the authors studied one-pass systems in which each job leaves the system after being processed at one processing step. Except for the network structure limitation, their works are actually more general than ours in the following two respects: (1) Job processing can depend on a stationary, random environment, and (2) their MaxWeight policies are a richer family of policies for a one-pass system. Although it has not been attempted here, it should be straightforward to generalize our results to stochastic processing networks with random environments. However, it is not at all clear how to generalize their general MaxWeight policies to our network setting.

Variants of maximum pressure and MaxWeight policies were first advanced by Tassiulas and Ephremides (1992) under different names for scheduling a multihop radio network. Their work was further studied by various authors for systems in different applications (Andrews et al. 2004, Dai and Prabhakar 2000, McKeown et al. 1999, Stolyar 2004, Tassiulas 1995, Tassiulas and Bhattacharya 2000, Tassiulas and Ephremides 1993), all limited to one-pass systems except Tassiulas and Bhattacharya (2000). The work of Tassiulas and Bhattacharya (2000) represents a significant advance in finding efficient operational policies for a wide class of networks, and is closely related to our current work. Although we were ignorant of their work when our work was performed, there is a significant amount of overlap and difference between these two works. The following are the major contrasts of these two papers. (a) They model server interdependence by directly imposing constraints on servers, whereas we use processing activities and constraints on them to model server interdependence; the latter is a more general approach to model server interdependence. (b) To the best of our knowledge, their model, when translated into our stochastic processing network framework, is a special network within a class of *strictly unitary networks*. In a latter network, each activity requires a *single server* that processes jobs in a *single buffer*. Thus, their model cannot model activities that require simultaneous possession of multiple servers nor activities that can simultaneously process jobs from multiple buffers. In particular, we are not able to see how their model can model operator-machine interactions and networks of data switches (see §9.2). (c) Their model requires processing speeds to depend only on buffers, not on activities. This assumption rules out many models like skill-based routing in call-center environments. (d) Our exogenous arrival model is more general. This generality allows us to model alternate routes at source levels (see §9.1). As a consequence of these model differences, they can focus on non-processor-splitting, nonpreemptive policies without additional assumptions on network structures. To prove that maximum pressure policies are throughput optimal for our general model, we need to allow processor splitting and preemption in our policies and to search for new assumptions on network structure (see the EAA Assumption in §4).

The maximum pressure policies are attractive in that their implementation uses minimal state information of the network. The deployment of a processor is decided based on the queue lengths in its serviceable buffers and the queue lengths at their immediately downstream buffers.

In particular, the decision does not use arrival rate information that is often hard or impossible to estimate reliably. The maximum pressure policies are not completely local in that they use immediately downstream buffer information of a processor. Using such information is not an issue in many manufacturing systems, but may be a problem for other systems. Searching for a purely local policy that is throughput optimal remains an open problem. See §9.1 for more discussions on this point.

In §2, we define the stochastic processing networks studied in this paper. In §3, we define the maximum pressure when processor splitting and preemption are allowed. We also define our notion of stability for a stochastic processing network. Section 4 introduces the static planning problem LP that was first introduced by Harrison (2000). The LP is proved to be necessary for the network to be stabilizable under some service policy (Theorem 1). We also introduce the extreme-allocation-available (EAA) assumption. The main theorem of this paper is also stated in this section (Theorem 2). In §5, we first introduce fluid models, a standard tool for proving network stability. We then outline a proof of Theorem 2, with supporting results delayed to Appendix A. In §6, we show that the EAA assumption is satisfied for strict Leontief networks that can model a wide class of systems, including networks of data switches. We also present an example where EAA is not satisfied. In §7, we first present an example showing that Harrison's LP is not sufficient for the stability of a stochastic processing network when processor splitting is *not* allowed. We adapt the maximum pressure policies to the non-processor-splitting setting, and show that non-processor-splitting maximum pressure policies are throughput optimal among non-processor-splitting policies. Section 8 shows that, in a reversed Leontief network, a nonpreemptive version of a maximum pressure policy is well defined and is throughput optimal. In §9, we show that our maximum pressure policies can be used to control multiclass queueing networks with alternate routes and networks of data switches.

2. Stochastic Processing Networks

In this section, we describe a variant of the class of stochastic processing networks advanced in Harrison (2000). The network is assumed to have $\mathbf{I}+1$ buffers, $\mathbf{J}$ activities, and $\mathbf{K}$ processors. Buffers, activities and processors are indexed by $i=0,\ldots,\mathbf{I}$, $j=1,\ldots,\mathbf{J}$, and $k=1,\ldots,\mathbf{K}$, respectively. For notational convenience, we define $\mathcal{I}=\{1,\ldots,\mathbf{I}\}$ the set of buffers excluding Buffer 0, $\mathcal{J}=\{1,\ldots,\mathbf{J}\}$ the set of activities, and $\mathcal{K}=\{1,\ldots,\mathbf{K}\}$ the set of processors. Each buffer, with infinite capacity, holds jobs or materials that await service. Buffer 0 is a special one that is used to model the outside world, where an infinite number of jobs await. Each activity can simultaneously process jobs from a set of buffers. It may require simultaneous possession of multiple processors to be active. Jobs departing from a buffer will go next to other buffers with certain probabilities that depend on the current activity taken.

2.1. Resource Consumption

Each activity needs one or more processors available to be active. For activity j, $A_{kj}=1$, if activity j requires processor k, and $A_{kj}=0$ otherwise. The $\mathbf{K}\times\mathbf{J}$ matrix $A=(A_{kj})$ is the resource consumption matrix. Each activity may be allowed to process jobs in multiple buffers simultaneously. For activity j, we use the indicator function B_{ji} to record whether buffer i can be processed by activity j. ($B_{ji}=1$ if activity j processes buffer i job.) The set of buffers i with $B_{ji}=1$ is said to be the *constituency* of activity j. It is denoted by $\mathcal{B}_j$. The constituency is assumed to be nonempty for each activity $j\in\mathcal{J}$, and may contain more than one buffer. For activity j, we use $\eta_j(\ell)$ to denote the ℓth activity j processing requirement. When the processing requirement is met, a job departs from each one of the constituent buffers. Allowing service times to depend on activities, not just buffers, creates an enormous amount of modeling flexibility, e.g., modeling cross-trained nonhomogeneous servers and multiple service modes of a server.

An activity j is said to be an *input activity* if it processes jobs only from Buffer 0; i.e., $\mathcal{B}_j=\{0\}$. An activity j is said to be a *service activity* if it does not process any job from Buffer 0; i.e., $0\notin\mathcal{B}_j$. We assume that each activity is either an input activity or a service activity. We further assume that each processor processes either input activities only or service activities only. A processor that only processes input activities is called an *input processor*, and a processor that only processes service activities is called an *service processor*. The input processors process jobs from Buffer 0 (outside) and generate the arrivals for the network. Introducing input processors allows us to model dynamic routing of arrivals as in Kelly and Laws (1993); see §9.1 for an example.

2.2. Routing

Buffer i jobs, after being processed by activity j, will go next to other buffers or leave the system. Among the first ℓ buffer i jobs that are processed by activity j, we use a nonnegative integer $\Phi^j_{ii'}(\ell)$ to denote the number of jobs that will go next to buffer i'. (If activity j does not process buffer i, $\Phi^j_{ii'}(\ell)=0$ for all i' and ℓ.) We use Φ^j to denote the routing process $\{\Phi^j_{ii'}(\ell)\colon i,i'\in\mathcal{I}\cup\{0\},\ \ell\geqslant 1\}$ associated with activity j. We assume that $\Phi^j_{00}(\ell)=0$ for each $\ell\geqslant 1$, and

$$\sum_{i'\in\mathcal{I}\cup\{0\}}\Phi^j_{ii'}(\ell)=\ell\quad\text{for each }\ell\geqslant 1\text{ and }i\in\mathcal{I}\cup\{0\}.$$

Using an activity-dependent routing process, we can model processing networks with alternate routes.

2.3. Resource Allocations

Because multiple activities may require usage of the same processor, not all activities can be simultaneously undertaken at a 100% level. For most of this paper, we assume

that each processor's service capacity is infinitely divisible, and processor splitting of a processor's service capacity is realizable. We use a nonnegative variable a_j to denote the level at which processing activity j is undertaken. When $a_j = 1$, activity j is employed at a 100% level. When $a_j = 0$, activity j is not employed. Suppose that the engagement level of activity j is a_j, with $0 \leqslant a_j \leqslant 1$. The processing requirement of an activity j job is depleted at rate a_j. (The job finishes processing when its processing requirement reaches 0.) The activity consumes $a_j A_{kj}$ fraction of processor k's service capacity per unit time. The remaining service capacity, $1 - a_j A_{kj}$, can be used for other activities. In many applications, notably in manufacturing systems, processor splitting is either impossible or not desirable, and thus the engagement level of an activity is either 0% or 100%. In §7, we will discuss an alternative model where processor sharing is prohibited.

We use $a = (a_j) \in \mathbb{R}_+^{\mathbf{J}}$ to denote the corresponding $\mathbf{J}$-dimensional allocation (column) vector, where $\mathbb{R}_+$ denotes the set of nonnegative real numbers. Because each processor k can decrease processing requirements at the rate of at most one per unit of time, we have

$$\sum_{j \in \mathcal{J}} A_{kj} a_j \leqslant 1 \quad \text{for each processor } k. \tag{1}$$

In vector form, $Aa \leqslant e$, where e is the $\mathbf{K}$-dimensional vector of ones. We assume that there is at least one input activity and that the input processors are never idle. Namely,

$$\sum_{j \in \mathcal{J}} A_{kj} a_j = 1 \quad \text{for each input processor } k. \tag{2}$$

We use $\mathcal{A}$ to denote the set of allocations $a \in \mathbb{R}_+^{\mathbf{J}}$ that satisfy (1) and (2).

Each $a \in \mathcal{A}$ represents an allowable allocation of the processors working on various activities. We note that $\mathcal{A}$ is bounded and convex. Let $\mathcal{E} = \{a^1, \ldots, a^{\mathbf{E}}\}$ be the set of extreme points of $\mathcal{A}$, where the total number $\mathbf{E}$ of extreme points is finite.

2.4. Service Policies

Each job in a buffer is assumed to be processed by one activity in its entire stay at the buffer. A processing of an activity can be preempted. In this case, each in-service job is "frozen" by the activity. The next time the activity is made active again, the processing is resumed from where it was left off. In addition to the availability of processors, a (nonpreempted) activity can be made active only when each constituent buffer has jobs that are not in service or frozen. We assume that, within each buffer, *head-of-line* policy is used. When a (nonpreempted) activity becomes active with a given engagement level, the leading job in each buffer that is not in service or frozen is processed. If multiple activities are actively working on a buffer, there are multiple jobs in the buffer that are in service. For an allocation a, if there is an activity j with $a_j > 0$ that cannot be made active, the allocation is infeasible. At any given time t, we use $\mathcal{A}(t)$ to denote the set of allocations that are *feasible* at that time.

The set of feasible allocations $\mathcal{A}(t)$ depends very much on whether preemption is allowed for the processing of activities. If preemption is not allowed, at any given time t, any incomplete activities must stay in service, thus reducing the set of feasible allocations at time t. If preemption is allowed, when an activity finishes its service and a new allocation is to be chosen, each activity that is not in the new allocation is preempted. Generally, our results fail to hold when preemption is not allowed. In §8, we will show that our results continue to hold for reversed Leontief networks when processing of activities is nonpreemptive.

2.5. Primitive Processes

The processing time sequences $\eta_j = \{\eta_j(\ell),\ \ell \geqslant 1\}$ for $j \in \mathcal{J}$ and the routing processes $\Phi^j = \{\Phi^j_{ii'}(\ell)\colon\ i, i' \in \mathcal{I} \cup \{0\},\ \ell \geqslant 1\}$ for $j \in \mathcal{J}$ are said to be the primitive processes of a stochastic processing network. Given a realization of the primitive processes and a service policy (including a tie-breaking policy for possible simultaneous arrivals), the network dynamics are completely determined.

We make a minimal assumption on the primitive processes (η, Φ). It basically assumes that processing rates and routing probabilities are well defined. We assume that for each activity $j \in \mathcal{J}$, there exist a nonnegative constant m_j and a nonnegative $(\mathbf{I}+\mathbf{1}) \times (\mathbf{I}+\mathbf{1})$ matrix P^j such that, with probability one,

$$\lim_{n \to \infty} \sum_{\ell=1}^{n} \eta_j(\ell)/n = m_j, \tag{3}$$

$$\lim_{\ell \to \infty} \Phi^j(\ell)/\ell = P^j. \tag{4}$$

Setting $\mu_j = 1/m_j$, one interprets μ_j as the processing rate of activity j. The matrix P^j is called the routing matrix for activity j.

3. The Maximum Pressure Service Policies

In this section, we define a family of policies, called *maximum pressure service policies*. We then introduce a notion of stability for a stochastic processing network. Under a mild assumption on network structure, we will prove in §5 that a maximum pressure policy is throughput optimal in the sense that it stabilizes a stochastic processing network if the network is stabilizable at all. For each buffer $i \in \mathcal{I}$, let buffer level $Z_i(t)$ denote the number of jobs in buffer i at time t. We use $Z(t)$ to denote the corresponding $\mathbf{I}$-dimensional column vector; we refer to $Z = \{Z(t),\ t \geqslant 0\}$ as the buffer-level process.

For each buffer $i = 1, \ldots, \mathbf{I}$ and each activity $j = 1, \ldots, \mathbf{J}$ we define

$$R_{ij} = \mu_j \left(B_{ji} - \sum_{i' \in \mathcal{I} \cup \{0\}} B_{ji'} P^j_{i'i} \right). \tag{5}$$

The $\mathbf{I} \times \mathbf{J}$ matrix $R = (R_{ij})$ is called the input-output matrix in Harrison (2002). (Harrison took R as part of a model specification to allow more general modeling capability.) One interprets R_{ij} as the average amount of buffer i material consumed per unit of activity j, with a negative value being interpreted to mean that activity j is a net producer of material in buffer i.

For a resource allocation $a \in \mathcal{A}$ and a given $z \in \mathbb{R}^{\mathbf{I}}_+$, define the corresponding total network pressure to be

$$p(a, z) = z \cdot Ra, \tag{6}$$

where, for two vectors x and y, $x \cdot y = \sum_\ell x_\ell y_\ell$ denotes the inner product. Although we interpret z as the buffer-level vector in the network, its components do not have to be integers.

The general idea of a maximum pressure policy is to employ an allocation

$$a^* \in \arg\max_{a \in \mathcal{A}} p(a, Z(t)) \tag{7}$$

at any given time t. Unfortunately, such an a^* is not always a feasible allocation. Note that $p(a, Z(t))$ is linear in a. Thus, the maximum in (7) is achieved at one of those extreme allocations. Namely, $p(a^*, Z(t)) = \max_{a \in \mathcal{E}} p(a, Z(t))$, where, as before, $\mathcal{E}$ is the set of extreme allocations of $\mathcal{A}$.

Recall that $\mathcal{A}(t)$ is the set of feasible allocations at time t. Namely, $\mathcal{A}(t)$ is the set of allocations $a = (a_j)$ such that at time t for each nonpreempted activity j with $a_j > 0$, the constituent buffers (those buffers i with $B_{ji} = 1$) have "fresh" jobs that are neither in service nor preempted. Define $\mathcal{E}(t) = \mathcal{E} \cap \mathcal{A}(t)$ to be the set of feasible extreme allocations at time t. Because preemption is assumed, one can argue that $\mathcal{E}(t)$ is always nonempty. For example, any extreme allocation that forces all service processors to stay idle is an element in $\mathcal{E}(t)$.

Definition 1. A service policy is said to be a *maximum pressure policy* if, at each time t, the network chooses an allocation $a^* \in \arg\max_{a \in \mathcal{E}(t)} p(a, Z(t))$.

When more than one allocation attains the maximum pressure, a tie-breaking rule is used. Our results are not affected by how ties are broken. However, for concreteness, one can order the extreme allocation set $\mathcal{E}$, and always choose the smallest, maximum pressure allocation.

Note that the buffer-level process Z does not change between processing completions. Thus, under a maximum pressure policy, allocations will not change between these completions. Each allocation decision is triggered by the completion of either an input activity or a service activity. Let $t_0 = 0$, and $\{t_n\colon n = 1, \ldots\}$ be the sequence of decision times under a maximum pressure policy. At decision time t_n, one observes the buffer level $Z(t_n)$, and chooses an allocation $a^n = f(Z(t_n))$, where $f\colon \mathbb{R}^{\mathbf{I}}_+ \to \mathcal{E} \subset \mathcal{A}$ is a function such that $f(z)$ maximizes $p(a, z)$ among all feasible allocations $a \in \mathcal{E}$ for each $z \in \mathbb{R}^{\mathbf{I}}_+$. The allocation remains fixed until the next activity completion time t_{n+1}.

A maximum pressure policy is an example of a stationary Markovian policy, with the buffer level $Z(t)$ serving as the state at time t. For a general stationary Markovian policy, a state $X(t)$ at time t can contain much more information, and the function f mapping from the state space to the allocation space $\mathcal{A}$ is general. For example, a state can contain the order in which jobs are lined up in each buffer, the remaining times of current activities in processing or preempted activities, and the time that each job has spent in the current buffer. Because a formal definition of a state is not essential to this paper, we do not attempt a precise definition here. Readers are referred to Bramson (1998) and Williams (1998) for an analogous, detailed treatment in the multiclass queueing network setting. By a *general service policy* we mean a policy under which the allocation a^n chosen at the nth decision time t_n is given by $a^n = f_n(X(t_0), X(t_1), \ldots, X(t_n)) \in \mathcal{A}(t_n)$ for some function f_n. Furthermore, f_n can be randomized in an arbitrary fashion.

We end this section by defining the pathwise stability for stochastic processing networks.

Definition 2. A stochastic processing network operating under a general service policy is said to be *pathwise stable* or simply *stable* if for every initial state, with probability one,

$$\lim_{t \to \infty} Z_i(t)/t = 0, \quad i \in \mathcal{I}. \tag{8}$$

Pathwise stability ensures that the total departure rate from the network is equal to the total input rate to the network. An unstable network incurs linear buildup of jobs in the system, at least for some network realizations. Although it will not be discussed further in this paper, one can employ other definitions of stability like positive Harris recurrence under some stronger assumptions on the primitive processes. Readers are referred to Dai (1995), Dai and Meyn (1995), and Stolyar (1995) for such possible extensions.

The central focus of this paper is to answer the following questions: (1) What is the natural condition on the primitive processes under which the stochastic processing network is stabilizable under some service policy, and (2) given that the network is stabilizable, are there any dynamic policies that use "minimal system information" and stabilize the network?

4. The Static Planning Problem and the Main Theorems

In this section, we state the main theorems of this paper. We first introduce an LP that will be used in the theorems to characterize the stability of a stochastic processing network. The LP, called the static planning problem in Harrison (2000), is defined as

$$\min \ \rho \tag{9}$$

$$\text{s.t. } Rx = 0, \tag{10}$$

$$\sum_{j\in\mathcal{J}} A_{kj}x_j \leqslant \rho \quad \text{for each service processor } k, \tag{11}$$

$$\sum_{j\in\mathcal{J}} A_{kj}x_j = 1 \quad \text{for each input processor } k, \tag{12}$$

$$x \geqslant 0. \tag{13}$$

For each activity $j \in \mathcal{J}$, x_j can be interpreted as the long-run fraction of time that activity j is active. With this interpretation, the left side of (10) is interpreted as the long-run *net flow rate* from the buffers. Equality (10) demands that, for each buffer, the long-run input rate to the buffer is equal to the long-run output rate from the buffer.

The following theorem provides a partial answer to question (1) at the end of §3.

THEOREM 1. *The static planning problem* (9)–(13) *has a feasible solution with* $\rho \leqslant 1$ *if the network is stable under some service policy.*

We leave the proof of the theorem to Appendix B. The next theorem, the main theorem of this paper, provides a complete answer to questions (1) and (2). To state the theorem, we need to introduce an assumption on the network structure under which maximum pressure policies are shown to be throughput optimal. For an allocation $a \in \mathcal{A}$, buffer i is said to be a constituent buffer of a if it can generate positive flow under a; i.e., $\sum_{j\in\mathcal{J}} a_j B_{ji} > 0$.

ASSUMPTION 1 (EAA ASSUMPTION). *For any buffer-level vector* $z \in \mathbb{R}_+^{\mathbf{I}}$, *there exists an extreme allocation* $a^* \in \mathcal{E}$ *that maximizes the network pressure* $p(a, z)$, *i.e.,* $p(a^*, z) = \max_{a\in\mathcal{E}} p(a, z)$, *and that for each constituent buffer* i *of* a^*, *the buffer level* z_i *is positive.*

The above assumption is called the extreme-allocation-available (EAA) assumption. It basically ensures that the maximum pressure allocation in (7) can be achieved by some feasible extreme allocation, i.e., $\max_{a\in\mathcal{A}} p(a, Z(t)) = \max_{a\in\mathcal{E}(t)} p(a, Z(t))$, when each nonempty buffer has sufficiently many jobs. The EAA Assumption is satisfied for a wide range of familiar networks, including strict Leontief networks introduced in Bramson and Williams (2004) and networks of switches (see §9.2). Assumption 1 fails to hold for some networks. In §6, we will discuss the assumption in more detail.

THEOREM 2. *Consider a stochastic processing network that satisfies Assumption* 1. *The network operating under a preemptive, processor-splitting maximum pressure policy is pathwise stable if the static planning problem* (9)–(13) *has a feasible solution with* $\rho \leqslant 1$.

The proof of Theorem 2 will be given in §5, with some of the supporting results proved in Appendix A. The maximum pressure policy can be generalized in the following way. For each buffer i, let γ_i be a positive number and θ_i be a real number. Given parameters $\gamma = (\gamma_i)$ and $\theta = (\theta_i)$, define the new network pressure at time t under allocation a to be $p(a, \tilde{Z}(t))$, where $\tilde{Z}_i(t) = \gamma_i Z_i(t) - \theta_i$. The parameterized maximum pressure policies associated with parameters γ and θ can be defined through the total network pressure as before. For the parameterized maximum pressure policies, we have the following corollary.

COROLLARY 1. *Consider a stochastic processing network that satisfies Assumption* 1. *The network operating under a parameterized, preemptive, processor-splitting maximum pressure policy is pathwise stable if the static planning problem* (9)–(13) *has a feasible solution with* $\rho \leqslant 1$.

Because the proofs of Theorem 2 and Corollary 1 are identical, to keep notation simple in the rest of this paper, we consider only the maximum pressure policies defined in Definition 1.

5. Fluid Models and an Outline of the Proof of Theorem 2

To prove Theorems 1 and 2, we adopt the standard fluid model approach (Dai 1999). In addition to introducing fluid models and their stability, this section outlines a proof of Theorem 2. The outline provides some key insights as to why a maximum pressure policy can stabilize a stochastic processing network.

The fluid model of a stochastic processing network is the deterministic, continuous analog of the stochastic processing network. It is defined by the following equations:

$$\bar{Z}_i(t) = \bar{Z}_i(0) + \sum_{i'\in\mathcal{J}\cup\{0\}} \sum_{j\in\mathcal{J}} \bar{T}_j(t)\mu_j B_{ji'} P^j_{i'i} - \sum_{j\in\mathcal{J}} \bar{T}_j(t)\mu_j B_{ji} \quad \text{for each } t \geqslant 0 \text{ and } i \in \mathcal{J}, \tag{14}$$

$$\bar{Z}_i(t) \geqslant 0 \quad \text{for each } t \geqslant 0 \text{ and } i \in \mathcal{J}, \tag{15}$$

$$\sum_{j\in\mathcal{J}} A_{kj}\big(\bar{T}_j(t) - \bar{T}_j(s)\big) = t - s \quad \text{for each } 0 \leqslant s \leqslant t \text{ and each input processor } k, \tag{16}$$

$$\sum_{j\in\mathcal{J}} A_{kj}\big(\bar{T}_j(t) - \bar{T}_j(s)\big) \leqslant t - s \quad \text{for each } 0 \leqslant s \leqslant t \text{ and each processor } k, \tag{17}$$

$$\bar{T} \text{ is nondecreasing and } \bar{T}(0) = 0. \tag{18}$$

Equations (14)–(18) are analogous to stochastic processing network equations (39)–(43) in Appendix A.1. They define the fluid model under any given service policy. Any quantity $(\bar{Z}, \bar{T})$ that satisfies (14)–(18) is a *fluid model solution*

to the fluid model that operates under a general service policy. Following its stochastic processing network counterparts to be discussed in Appendix A.1, each fluid model solution $(\bar{Z}, \bar{T})$ has the following interpretations: $\bar{Z}_j(t)$ the fluid level in buffer i at time t, and $\bar{T}_j(t)$ the cumulative amount of activity j processing time in $[0, t]$.

For each fluid model solution $(\bar{Z}, \bar{T})$, it follows from equations (16)–(17) that $\bar{T}$, and hence $\bar{Z}$, is Lipschitz continuous. Thus, the solution is absolutely continuous and has derivatives almost sure everywhere with respect to Lebesgue measure on $[0, \infty)$. A time $t > 0$ is said to be a regular point of the fluid model solution if the solution is differentiable at time t. For a function $f\colon \mathbb{R}_+ \to \mathbb{R}^d$ for some positive integer d, we use $\dot{f}(t)$ to denote the derivative of f at time t when the derivative exists. From (16)–(17), one has $\dot{\bar{T}}(t) \in \mathcal{A}$ at each regular time t. Thus, for a fluid model solution $(\bar{Z}, \bar{T})$ under a general service policy, the network pressure $R\dot{\bar{T}}(t) \cdot \bar{Z}(t)$ under allocation $\dot{\bar{T}}(t)$ when the fluid level is $\bar{Z}(t)$ is less than or equal to the maximum pressure $\max_{a\in\mathcal{A}} Ra \cdot \bar{Z}(t)$. Namely,

$$R\dot{\bar{T}}(t) \cdot \bar{Z}(t) \leqslant \max_{a\in\mathcal{A}} Ra \cdot \bar{Z}(t) = \max_{a\in\mathcal{E}} Ra \cdot \bar{Z}(t). \tag{19}$$

Under a specific service policy, there are additional fluid model equations. Under a maximum pressure policy and the EAA Assumption,

$$R\dot{\bar{T}}(t) \cdot \bar{Z}(t) = \max_{a\in\mathcal{E}} Ra \cdot \bar{Z}(t) \tag{20}$$

for each regular time t. Thus, under a maximum pressure policy, the instantaneous activity allocation $\dot{\bar{T}}(t)$ in the fluid model maximizes the network pressure at time t. Each fluid model equation will be justified through a fluid limit procedure as described in Appendix A.2. In particular, the fluid model equation (20) will be justified in Lemma 4 in Appendix A.3.

We now give another interpretation of a maximum pressure policy. Let $(\bar{Z}, \bar{T})$ be a fluid model solution. Consider the following quadratic Lyapunov function:

$$f(t) = \sum_i \bar{Z}_i^2(t). \tag{21}$$

At a regular time t,

$$\dot{f}(t) = 2\dot{\bar{Z}}(t) \cdot \bar{Z}(t) = -2R\dot{\bar{T}}(t) \cdot \bar{Z}(t), \tag{22}$$

where, in the second equality, we have used the vector form of fluid model equation (14),

$$\bar{Z}(t) = \bar{Z}(0) - R\bar{T}(t). \tag{23}$$

It follows from (19), (20), and (22) that the derivative of "system energy" $f(t)$ is minimized when $\dot{\bar{T}}(t)$ is chosen as a fluid model solution under a maximum pressure policy. Thus, a maximum pressure policy is *system greedy* in that the "system energy" decreases fastest (or increases slowest) at any regular time.

In addition to providing interpretations of a maximum pressure policy in the fluid model setting, the fluid model allows us to prove our main theorems. The following theorem provides a connection between the stability of a stochastic processing network and the weak stability of the corresponding fluid model, a notion we first define now.

Definition 3. A fluid model is said to be *weakly stable* if for every fluid model solution $(\bar{Z}, \bar{T})$ with $\bar{Z}(0) = 0$, $\bar{Z}(t) = 0$ for $t \geqslant 0$.

Theorem 3. *For a stochastic processing network, if the corresponding fluid model is weakly stable, it is pathwise stable.*

The proof of Theorem 3 will be presented in Appendix A.2. In light of the theorem, the following theorem provides a complete proof of Theorem 2.

Theorem 4. *Assume that the LP* (9)–(13) *has a feasible solution with* $\rho \leqslant 1$ *and the EAA Assumption is satisfied. The fluid model under a maximum pressure policy is weakly stable.*

Proof. Let $(\bar{Z}, \bar{T})$ be a fluid model solution with $\bar{Z}(0) = 0$, and f be the quadratic Lyapunov function as defined in (21). Let x^* be a solution to the LP (9)–(12) with $\rho \leqslant 1$. Obviously, $x^* \in \mathcal{A}$ because $\rho \leqslant 1$. Moreover $Rx^* = 0$. It follows from (22) and (20) that

$$\begin{aligned}\dot{f}(t) &= -2R\dot{\bar{T}}(t) \cdot \bar{Z}(t)\\ &= -2\max_{a\in\mathcal{A}} Ra \cdot \bar{Z}(t)\\ &\leqslant -2Rx^* \cdot \bar{Z}(t)\\ &= 0\end{aligned}$$

for each regular time t. Because $f(0) = 0$, we have $f(t) = 0$ for $t \geqslant 0$, proving that the fluid model is weakly stable. □

We end this section by stating a stronger version of Theorem 4. This result is of independent interest. It will not be used in the rest of this paper. The proof can be found in Appendix B. We first make another minor assumption on the stochastic processing network.

Assumption 2. *There exists an* $x \geqslant 0$ *such that* $Rx > 0$.

Definition 4. A fluid model is said to be *stable* if there exists a constant $\delta > 0$ such that for every fluid model solution $(\bar{Z}, \bar{T})$ with $|\bar{Z}(0)| \leqslant 1$, $\bar{Z}(t) = 0$ for $t \geqslant \delta$.

Theorem 5. *For a stochastic processing network satisfying Assumptions* 1 *and* 2 *and operating under a preemptive processor-splitting maximum pressure policy, the corresponding fluid model is stable if the LP* (9)–(12) *has a feasible solution with* $\rho < 1$.

6. The EAA Assumption Revisited

In this section, we introduce *strict Leontief networks* and verify that the extreme-allocation-available (EAA) Assumption, Assumption 1, is always satisfied for such networks. We then present a network example for which the EAA Assumption fails to satisfy.

6.1. Strict Leontief Networks

A stochastic processing network is said to be *strict Leontief* if each service activity is associated with exactly one buffer; i.e., $\mathcal{B}_j$ contains exactly one buffer for each activity $j \in \mathcal{J}$. (Recall that each input activity j is assumed to be associated with Buffer 0 only; i.e., $\mathcal{B}_j = \{0\}$.)

THEOREM 6. *Assumption* 1 *is satisfied for strict Leontief networks.*

PROOF. For strict Leontief networks, each row j of matrix B has exactly one entry equal to 1. Denote the corresponding buffer as $i(j)$. Then, $B_{ji} = 0$ and $R_{ij} \leqslant 0$ for each $i \in \mathcal{I}$ and $j \in \mathcal{J}$, such that $i \neq i(j)$. For any vector $z \in \mathbb{R}_+^{\mathbf{I}}$, we define $\mathcal{J}_0$ as the set of service activities j with $z_{i(j)} = 0$. It is sufficient to show that there exists an allocation $a^* \in \arg\max_{a \in \mathcal{E}} z'Ra$ with $a_j^* = 0$ for all $j \in \mathcal{J}_0$. Let $\hat{a}$ be any extreme allocation such that $z'R\hat{a} = \max z'Ra$. Define $\tilde{a}$ via

$$\tilde{a}_j = \begin{cases} 0, & j \in \mathcal{J}_0, \\ \hat{a}_j, & j \notin \mathcal{J}_0. \end{cases}$$

Obviously, $\tilde{a} \in \mathcal{A}$. Note that

$$z'R\hat{a} = \sum_i \sum_j z_i R_{ij} \hat{a}_j = \sum_i \sum_{j \in \mathcal{J}_0} z_i R_{ij} \hat{a}_j + \sum_i \sum_{j \in \mathcal{J} \setminus \mathcal{J}_0} z_i R_{ij} \hat{a}_j$$
$$= \sum_i \sum_{j \in \mathcal{J}_0} z_i R_{ij} \hat{a}_j + z'R\tilde{a}.$$

Because

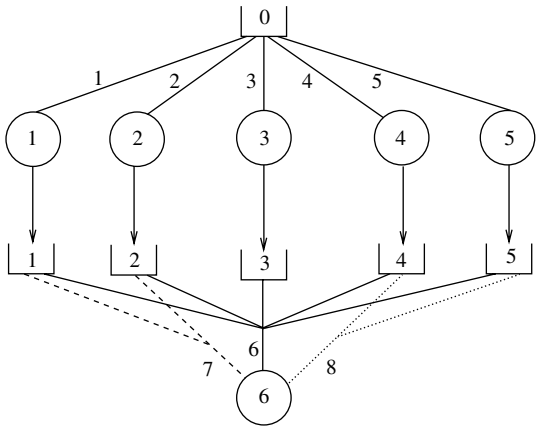
$$\sum_i \sum_{j \in J_0} z_i R_{ij} \hat{a}_j = \sum_{j \in J_0} \sum_{i \neq i(j)} z_i R_{ij} \hat{a}_j \leqslant 0,$$

we have $z'R\tilde{a} \geqslant z'R\hat{a}$. Because $z'R\hat{a} = \max_{a \in \mathcal{A}} z'Ra$, we have $z'R\tilde{a} = \max_{a \in \mathcal{A}} z'Ra$. If $\tilde{a}$ is an extreme allocation, then the proof is complete. Otherwise, it is a convex combination of some extreme allocations. Choose any one of these allocations as a^*. Then, $z'Ra^* = \max_{a \in \mathcal{A}} z'Ra$, and $a_j^* = 0$ for all $j \in \mathcal{J}_0$. □

6.2. A Network Example Not Satisfying the EAA Assumption

In this section, we present a network example for which the EAA Assumption is not satisfied. The network has a feasible solution with $\rho < 1$ to the static planning problem (9)–(13), yet it is not stable under any maximum pressure policy. The network is shown to be stable under some allocation policy.

Figure 1. An example that does not satisfy the EAA Assumption.



Consider the network depicted in Figure 1. The network has five internal buffers, in addition to the external Buffer 0. Buffers are represented by open rectangular boxes. There are six processors represented by circles, and eight activities labeled on the lines connecting circles to buffers. All processors, except Processor 6, are input processors. For $\ell = 1, \ldots, 5$, input processor ℓ works on activity ℓ to generate the arrivals to buffer ℓ with rate μ_ℓ. Each time, the service processor, Processor 6, can employ one activity from Activities 6, 7, and 8. By employing Activity 6, the service processor processes jobs from all five buffers simultaneously with rate μ_6. By employing Activity 7 (or 8), the service processor works simultaneously on Buffers 1 and 2 (or 4 and 5) with rate μ_7 (or μ_8). It is clear that all input activities must always be employed. However, each time the sum of allocations to all processing activities cannot exceed 1. Therefore, there are a total of four extreme allocations. They are $a^1 = (1, 1, 1, 1, 1, 1, 0, 0)'$, $a^2 = (1, 1, 1, 1, 1, 0, 1, 0)'$, $a^3 = (1, 1, 1, 1, 1, 0, 0, 1)'$, and $a^4 = (1, 1, 1, 1, 1, 0, 0, 0)'$.

We assume that all the processing times are deterministic. Their values are specified as follows. For activities $j = 1, 2$, and 3, $\eta_j(\ell) = 2$, $\ell \geqslant 1$. For activities $j = 4$ and 5, $\eta_j(1) = 1$, and $\eta_j(\ell) = 2$ for $\ell \geqslant 2$. For activities $j = 6, 7$, and 8, $\eta_j(\ell) = 1$, $\ell \geqslant 1$. Therefore,

$$\mu_j = \begin{cases} 0.5, & j = 1, \ldots, 5, \\ 1, & j = 6, 7, \text{ and } 8. \end{cases}$$

The input-output matrix R in (5) can be written as

$$R = \begin{pmatrix} -0.5 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & -0.5 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & -0.5 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -0.5 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & -0.5 & 1 & 0 & 1 \end{pmatrix}.$$

It can be easily checked that $x = (1, 1, 1, 1, 1, 1/2, 0, 0)'$ and $\rho = 1/2$ is a feasible solution to the static allocation problem (9)–(13).

The system is unstable under any maximum pressure policy. In the following, we show that for any maximum pressure policy with given parameters (γ, θ), the queue size of Buffer 3 can grow without bound under a certain initial condition. Without loss of generality, we assume that $\gamma_i = 1$ for each buffer i. Now consider the system with initial buffer sizes $Z_1(0) = \theta_1 + \theta_2$, $Z_5(0) = \theta_4 + \theta_5$, and $Z_i(0) = 0$ for $i = 2$, 3, and 4. At time $t < 1$, Buffers 2, 3, and 4 are empty. Thus, none of the service activities can be employed, and the service processor is idle. At time $t = 1$, Buffers 4 and 5 each have an arrival. Thus, $Z_4(t) = 1$ and $Z_5(t) = \theta_4 + \theta_5 + 1 \geqslant 1$, whereas Buffers 2 and 3 remain empty. Therefore, allocations a^1 and a^2 are not feasible at time $t = 1$. Because Buffers 4 and 5 have jobs at time $t = 1$, allocation a^3 is feasible. Furthermore, one can check that the network pressure $p(a^3, Z(t))$ under allocation a^3 is strictly larger than the network pressure $p(a^4, Z(t))$ under allocation a^4. Under the maximum pressure policy, allocation a^3 will be employed at time $t = 1$. At time $t = 2$, Buffers 1, 2, and 3 each have an arrival, and Buffers 4 and 5 each have a departure. Thus, $Z_1(t) = \theta_1 + \theta_2 + 1 \geqslant 1$, $Z_2(t) = 1$, $Z_3(t) = 1$, and $Z_4(t) = 0$. Because $Z_4(t) = 0$, allocations a^1 and a^3 are not feasible. It is easy to verify that a^2 will be employed under the maximum pressure policy at time $t = 2$. At time $t = 3$, $Z_2(t) = 0$, $Z_3(t) = 1$, $Z_4(t) = 1$, and $Z_5(t) = \theta_4 + \theta_5 + 1 \geqslant 1$. One can argue similar to the reasoning at time $t = 1$ that allocation a^3 will be employed at time $t = 3$. At time $t = 4$, $Z_1(t) = \theta_1 + \theta_2 + 1 \geqslant 1$, $Z_2(t) = 1$, $Z_3(t) = 2$, and $Z_4(t) = 0$. One can argue similar to the reasoning at time $t = 2$ that allocation a^2 will be employed at time $t = 4$. Continuing this process, one realizes that at any time either Buffer 2 or 4 will be empty. Therefore, Activity 6 will always be infeasible, and thus allocation a^1 is never feasible. Hence, jobs in Buffer 3 will never be processed. The system is unstable.

So far, we have verified that our network example has a feasible solution to (9)–(13), yet none of the maximum pressure policies stabilize the network. These facts do not contradict Theorem 2 because the network does not satisfy the EAA Assumption, a claim we now verify. Letting

$$z_j = \begin{cases} 0, & j \neq 3, \\ 1, & j = 3. \end{cases}$$

It is easy to see that $a^1 = \arg\max_{a \in \mathscr{E}} z \cdot Ra$, and a^1 is the only one that achieves the maximum. However, $a^1_6 = 1$ and $B_{61} = 1$, whereas $z_1 = 0$. This violates the EAA Assumption.

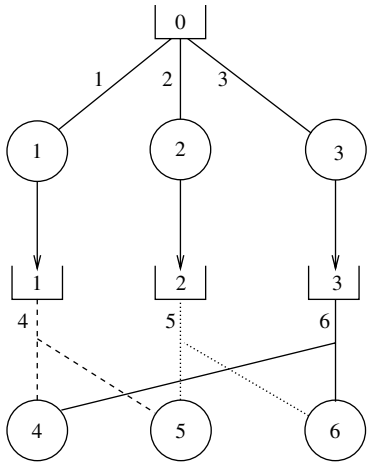
A closer examination of the instability analysis of the maximum pressure policies reveals that under a maximum pressure policy, allocations a^2 and a^3 are so "greedily" employed that Activity 6 never has a chance to be employed. If the service processor takes Activity 6 when all five buffers are nonempty and idles otherwise, the system would be stable. To see this, we first assume that the system is initially empty. It is easy to see that $Z_4(t) = Z_5(t) = 1$ for all $t \geqslant 1$, and $Z_1(t) = Z_2(t) = Z_3(t) = 0$ for $t \in [2n+1, 2n+2)$, and $Z_1(t) = Z_2(t) = Z_3(t) = 1$ for $t \in [2n+2, 2n+3)$. Thus, (8) holds. For more general initial conditions, one can check that $Z_i(t) \leqslant Z_i(0) + 1$ for $i = 1, \ldots, 5$. Thus, (8) holds in general, proving the stability of the network.

7. Non-Processor-Splitting Service Policies

In many applications including manufacturing systems, processor splitting is not allowed. In this case, each allocation a has components that are either 1 or 0; i.e., $a \in \mathbb{Z}_+^{\mathbf{J}}$, where $\mathbb{Z}_+$ denotes the set of nonnegative integers. Because there are fewer possible allocations to choose from at each decision point, the stability results developed earlier in the paper may not hold anymore. In this section, we first provide an example for which, unlike Theorem 2, Harrison's static planning problem (9)–(13) does not determine the stability region of a stochastic processing network when processor splitting is not allowed. We then establish a theorem that is analogous to Theorem 2 when processor splitting is not allowed. Finally, we introduce a class of networks, called reversed Leontief networks, for which the static planning problem (9)–(13) still determines the stability regions even when processor splitting is not allowed. As in previous sections, preemption is assumed throughout this section.

Consider the following example. As depicted in Figure 2, the network has four buffers (including Buffer 0 representing the outside world), six processors, and six activities. Buffers are represented by open rectangular boxes, processors are represented by circles, and activities are labeled on lines connecting buffers with processors. Processors 1, 2, and 3 are input processors, and Processors 4, 5, and 6 are service processors. Activities 1, 2, and 3 are input activities. Each input activity requires one input processor, taking input from Buffer 0 and producing output to

Figure 2. A processing network for which processor sharing is important.



the corresponding buffer, as indicated in the figure. Activities 4, 5, and 6 are service activities. Activity 4 requires Processors 4 and 5 simultaneously to process jobs from Buffer 1. Activity 5 requires Processors 5 and 6 simultaneously to process jobs from Buffer 2. Activity 6 requires Processors 4 and 6 simultaneously to process jobs from Buffer 3. The processing times of each activity are assumed to be deterministic. For each input activity i, the processing rate μ_i is assumed to be 0.4, $i = 1, 2, 3$. For each service activity i, the processing rate μ_i is assumed to be 1.0, $i = 4, 5, 6$. It follows from the definitions of input-output matrix R in (5) and the resource consumption matrix A that

$$R = \begin{pmatrix} -0.4 & 0 & 0 & 1 & 0 & 0 \\ 0 & -0.4 & 0 & 0 & 1 & 0 \\ 0 & 0 & -0.4 & 0 & 0 & 1 \end{pmatrix} \quad \text{and}$$

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix}.$$

It is easy to check that $x = (1, 1, 1, 0.4, 0.4, 0.4)'$ and $\rho = 0.8$ is the optimal solution to the static planning problem (9)–(13). However, when processor splitting is not allowed, we know that at any given time only one service activity can be active. Thus, the total maximum departure rate from the system is 1. However, the total rate into Buffers 1, 2, and 3 is 1.2, exceeding the total departure rate. Hence, the system is unstable.

Note that the popular round-robin mechanism as in Andradóttir et al. (2003) is a temporal way to share a processor's capacity. The above example cannot be stabilized by any temporal sharing of the processors. However, it is stable if each service processor is shared at the 50% level among its activities.

To develop an analogous stability theory in the non-processor-splitting case, we first change the allocation space $\mathscr{A}$ to $\mathscr{N}$, where $\mathscr{N}$ is the set of allocations $a \in \mathbb{Z}_+^{\mathbf{J}}$ that satisfy (1) and (2). Note that each allocation in $\mathscr{N}$ is an extreme one. Using $\mathscr{N}$ to replace $\mathscr{E}$, one can define a maximum pressure policy exactly as in §3. In the example described earlier in this section, $\mathscr{N} \neq \mathscr{E}$.

Consider the following static allocation problem (SAP): Find $\pi = (\pi_a)$ satisfying

$$Rx = 0, \tag{24}$$

$$x = \sum_{a \in \mathscr{N}} a\pi_a, \tag{25}$$

$$\sum_{a \in \mathscr{N}} \pi_a = 1, \tag{26}$$

$$\pi_a \geqslant 0 \quad \text{for each } a \in \mathscr{N}. \tag{27}$$

Here, π_a is interpreted as the long-run fraction of time that allocation a is employed, and x retains the same interpretation as in the static planning problem (10)–(13).

THEOREM 7. *The static allocation problem* (24)–(27) *has a feasible solution if the stochastic processing network is pathwise stable under some non-processor-splitting service policy.*

The proof of Theorem 7 is similar to the proof of Theorem 1. We will provide an outline of the proof in Appendix B. Theorem 7 says that the feasibility of the static allocation problem (24)–(27) is necessary for the network to be pathwise stable under any non-processor-splitting policy. The following theorem asserts that if the static allocation problem (24)–(27) is feasible, the network is pathwise stable under a non-processor-splitting maximum pressure policy. Thus, non-processor-splitting maximum pressure policies are throughput optimal among non-processor-splitting policies.

THEOREM 8. *Consider a stochastic processing network that satisfies Assumption* 1 *with* $\mathscr{E}$ *replaced by* $\mathscr{N}$. *If the static allocation problem* (24)–(27) *has a feasible solution, the network operating under a non-processor-splitting maximum pressure policy is pathwise stable.*

PROOF. First, under Assumption 1 with $\mathscr{E}$ being replaced by $\mathscr{N}$, the fluid model operating under a non-processor-splitting maximum pressure is defined by equations (14)–(18) and

$$R\dot{\bar{T}}(t) \cdot \bar{Z}(t) = \max_{a \in \mathscr{N}} Ra \cdot \bar{Z}(t). \tag{28}$$

The last equation replaces the fluid model equation (20) for the fluid model operating under a processor-splitting maximum pressure policy. With $\mathscr{N}$ replacing $\mathscr{A}$, Lemmas 4 and 5 still hold. They provide a justification of fluid model equations (14)–(18) and (28), via fluid limits introduced in Appendix A.2.

Let $\bar{\mathbb{X}} = (\bar{Z}, \bar{T})$ be a fluid model solution with $\bar{Z}(0) = 0$. Consider the quadratic Lyapunov function $f(t)$ defined in (21). It follows from (22) and (28) that, for any regular point t,

$$\begin{aligned} \dot{f}(t) &= -2R\dot{\bar{T}}(t) \cdot \bar{Z}(t) \\ &= -2 \max_{a \in \mathscr{N}} Ra \cdot \bar{Z}(t) \\ &\leqslant -2\left(R \sum_{a \in \mathscr{N}} a\pi_a \right) \cdot \bar{Z}(t) \end{aligned}$$

for any distribution $\pi = (\pi_a)_{a \in \mathscr{N}}$ with $\pi_a \geqslant 0$ and $\sum_{a \in \mathscr{N}} \pi_a = 1$. Let π be a feasible solution to (24)–(27). Then,

$$R \sum_{a \in \mathscr{N}} a\pi_a = 0$$

and $\sum_{a\in\mathcal{N}} \pi_a = 1$. Thus, $\dot{f}(t) \leqslant 0$. Hence, $\bar{Z}(t) = 0$ for $t \geqslant 0$, proving the weak stability of the fluid model. By Theorem 3, the stochastic processing network is pathwise stable. □

When $\mathcal{N} = \mathcal{E}$, the maximum pressure policies used in §3 are actually non-processor-splitting. Therefore, Theorem 2 still holds in this case.

Definition 5. A stochastic processing network is said to be *reversed Leontief* if each activity requires exactly one processor.

Lemma 1. *For a reversed Leontief network, every extreme allocation is an integer allocation*; *i.e.*, $\mathcal{E} = \mathcal{N}$.

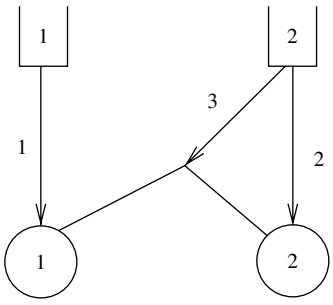
See Appendix B for the proof of Lemma 1. From the lemma, the maximum pressure policies in a reversed Leontief network are always throughput optimal whether processor splitting is allowed or not. The resulting allocations are always non-processor-splitting.

8. Nonpreemptive Service Policies

In the definition of maximum pressure policies, we have implicitly assumed that preemption of activities is allowed. These policies are defined through extreme allocations. When preemption is not allowed, at any given time the feasible set of allocations is reduced. It is possible that this feasible set does not contain any extreme allocations, yielding the current definition of the policy invalid in the nonpreemption case. When preemption is not allowed, even when a maximum pressure policy can be defined, the example to be presented below shows that Theorem 2 does not hold in general. In other words, even if the static planning problem (9)–(13) has a feasible solution, the network is not stable under a nonpreemption version of a maximum pressure policy. In this section, we will first present an example. We then prove that when a stochastic processing network has some special structure, a nonpreemptive maximum pressure service policy is well defined and is throughput optimal.

Consider the example depicted in Figure 3, with two service processors processing jobs in Buffers 1 and 2. Jobs arrive at Buffer 1 at time $0.5 + 2n$, $n = 0, 1, 2, \ldots$, and arrive at Buffer 2 at time $1 + 1.5n$, $n = 0, 1, 2, \ldots$. The network is assumed to be empty initially. There are three service activities: Activity 1 requires Service Processor 1 and processes jobs in Buffer 1 with a deterministic processing requirement of one unit of time; Activity 2 requires Service Processor 2 and processes jobs in Buffer 2 with a deterministic processing requirement of two units of time; and Activity 3 requires both Servers 1 and 2 and processes jobs in Buffer 2 with a deterministic processing requirement of one unit of time. (Input activities and Buffer 0 representing the outside world are not drawn in the figure.)

Figure 3. Nonpreemption could make a system unstable.



One can easily verify that the static planning problem (9)–(13) has a feasible solution with $\rho = 11/12$. Because each activity processes jobs from inside a single buffer, the network is strictly Leontief. By Theorem 6, the EAA Assumption is satisfied. Hence, any preemptive, processor-splitting maximum pressure policy is pathwise stable for the network. We now show that a nonpreemptive maximum pressure policy is not pathwise stable. At time $t = 0.5$, there is an arrival at Buffer 1, and Activity 1 becomes active because the allocation $(1, 0, 0)$ is the maximum pressure allocation. At time $t = 1$, there is an arrival at Buffer 2. Because Processor 1 has been assigned to Activity 1, which cannot be interrupted, only allocations $(1, 1, 0)$ and $(1, 0, 0)$ are feasible at $t = 1$. It can be verified that allocation $(1, 1, 0)$ is the maximum pressure allocation at this time. Therefore, Activity 2 becomes active at time $t = 1$. At time $t = 1.5$, Processor 1 completes the Activity 1 processing (of the job in Buffer 1). Because Processor 2 is still active processing Activity 2, the only feasible allocation during time interval $[1.5, 2.5]$ is $(0, 1, 0)$. At time $t = 2.5$, Activity 1 will be active again. It is easy to verify that Activity 1 will be active during time interval $[0.5 + 2n, 1.5 + 2n]$ and Activity 2 completes a processing at time $1 + 2n$, at which Processor 1 is tied with Activity 1 processing. Thus, Processors 1 and 2 have no chance to engage Activity 3. Therefore, under the nonpreemptive maximum pressure policy, the departure rate of Buffer 2 is $1/2$, which is less than the arrival rate $2/3$. Hence, the network is not pathwise stable.

Recall that a stochastic processing network is called a reversed Leontief network if each activity needs exactly one processor to be active. By Lemma 1, each maximum pressure policy is necessarily non-processor-splitting. We now show that, in a reversed Leontief network, the nonpreemptive version of a maximum pressure policy is well defined. Furthermore, the network operating under such a policy is throughput optimal.

The following lemma shows that the nonpreemption maximum pressure policies are well defined for reversed Leontief networks.

Lemma 2. *For reversed Leontief networks, the nonpreemption maximum pressure policies are well defined. That is,* $\mathcal{E}(t)$ *is nonempty for each time* $t \geqslant 0$.

PROOF. It is easy to see that there exists a feasible extreme allocation initially. For instance, one can let each input processor choose any associated activity and let all service processors idle. Now suppose that a nonpreemptive maximum pressure policy is well defined prior to time t. We consider two cases, depending on if there is a service activity completion at time t. (1) Assume that there is no service activity completion at time t. For this case, the previous allocation is still feasible and also extreme. Thus, $\mathscr{E}(t)$ is not empty. (2) Assume that there is a service activity completion at time t. Let a be the allocation immediately preceding t. By assumption, a is extreme. We now define a new allocation $\tilde{a}$. We let $\tilde{a}_j = a_j$ for all activities j except the service activities that have completions at time t. For these service activities j, we let $\tilde{a}_j = 0$. Clearly, $\tilde{a}$ is still extreme. Because each service activity is associated with exactly one processor, allocation $\tilde{a}$ is a feasible allocation. Thus, $\mathscr{E}(t)$ is not empty. □

In the rest of this section, we will prove a theorem that nonpreemption maximum pressure policies are throughput optimal in reversed Leontief networks. To state the theorem, we need a slightly stronger assumption on the service times than the one in (3).

ASSUMPTION 3. *For each activity $j \in \mathscr{J}$, there exist $\epsilon_j > 0$ and $m'_j > 0$ such that, with probability one,*

$$\lim_{n\to\infty} n^{-1} \sum_{\ell=1}^{n} \eta_j^{1+\epsilon_j}(\ell) = m'_j. \tag{29}$$

THEOREM 9. *Assume that Assumptions* 1 *and* 3 *are satisfied. Then, the reversed Leontief networks operating under nonpreemptive, non-processor-splitting maximum pressure policies are pathwise stable if the static planning problem* (10)–(13) *has a feasible solution* (x, ρ) *with* $\rho \leqslant 1$.

REMARK. In some networks, preempting service activities is allowed, but preempting input activities is not. In this case, if the input processors and input activities satisfy the reversed Leontief property, the stability result analogous to Theorem 9 still holds.

To prove Theorem 9, in light of the proof of Theorems 2 and 4, it is sufficient to prove that the fluid model for a reversed Leontief network operating under a nonpreemption maximum pressure policy is still defined by (14)–(18) and (20). See Appendix B for the proof. The proof needs the following lemma, which may be of independent interest.

To present the lemma, recall that in a reversed Leontief network, by Lemma 1, each extreme allocation a is an integer allocation. Thus, each processor k is employed by at most one activity. We use $j_k(a)$ to denote the activity that processor k is working on under allocation a. We set $j_k(a) = 0$ if processor k is idle under allocation a. Recall that $\mathscr{J}(k)$ is the set of possible activities that processor k can take, including the idle Activity 0.

For an activity $j \in \mathscr{J}$, define activity j pressure when the buffer level is $Z(t)$ to be

$$p(j, Z(t)) = \sum_{i\in\mathscr{I}} R_{ij} Z_i(t). \tag{30}$$

Clearly, the total network pressure $p(a, Z(t))$ under an allocation a is equal to

$$\sum_{j\in\mathscr{J}} a_j p(j, Z(t)).$$

If j is an idle activity, $p(0, t)$ is set to be 0. The following lemma shows that, in a reversed Leontief network, a maximum pressure policy yields a *separable policy* in the sense that when there are sufficiently many jobs in each constituent buffer, processor k makes decisions based on pressures $p(j, t)$, $j \in \mathscr{J}(k)$, independent of other processors.

LEMMA 3. *In a reversed Leontief network, for any allocation $a \in \mathscr{E}$, $p(a, Z(t)) = \max_{a'\in\mathscr{E}} p(a', Z(t))$ if and only if*

$$j_k(a) \in \arg\max_{j\in\mathscr{J}(k)} p(j, Z(t)) \quad \textit{for all } k \in \mathscr{K}.$$

PROOF. Suppose that there exist a processor k and an activity $j \in \mathscr{J}(k)$ such that $p(j_k(a), Z(t)) < p(j, Z(t))$. Then, we define another allocation $\tilde{a} = a - e_{j_k(a)} + e_j$, where, as before, e_j is the **J**-dimensional vector with the jth component equal to 1 and all other components 0. Clearly, $\tilde{a}$ is an integer, feasible allocation, and hence it is extreme. Moreover,

$$\begin{aligned} p(\tilde{a}, Z(t)) &= p(a, Z(t)) - p(j_k(a), Z(t)) + p(j, Z(t)) \\ &> p(a, Z(t)). \end{aligned}$$

Thus, a cannot be a maximum allocation.

Conversely, suppose that a is an extreme allocation that satisfies $p(j_k(a), Z(t)) \geqslant p(j, Z(t))$ for all $j \in \mathscr{J}(k)$ and all $k \in \mathscr{K}$. We would like to show that a is a maximum allocation. Let $\hat{a}$ be an extreme maximum allocation; i.e., $\max_{a'\in\mathscr{E}} p(a', Z(t)) = p(\hat{a}, Z(t))$. Then,

$$\begin{aligned} p(a, Z(t)) &= \sum_k p(j_k(a), Z(t)) \geqslant \sum_k p(j_k(\hat{a}), Z(t)) \\ &= p(\hat{a}, Z(t)), \end{aligned}$$

which implies that a is a maximum allocation. □

9. Applications

In this section, we describe two applications for which the non-processor-splitting, nonpreemptive maximum pressure policies are throughput optimal. The first application is to queueing networks with alternative routes. Such a network can be modeled as a stochastic processing network that is *strictly unitary*: Each activity is associated with exactly one buffer and it employs exactly one processor. The other application is to networks of data switches. The resulting stochastic processing networks are not unitary. To the best of our knowledge, our maximum pressure policies are the first family of stationary policies, with the buffer level as a state, that are proven to be throughput optimal in these two settings.

Recently, there have been a lot of research activities on parallel server systems, or, more generally, queueing networks with flexible servers (Andradóttir et al. 2003; Bell and Williams 2001, 2004; Gans and van Ryzin 1997; Harrison and López 1999; Squillante et al. 2001; Williams 2000). These networks have been used to model call centers with cross-trained operators (Armony 1999, Armony and Bambos 2003, Gans et al. 2003). We will not get into detailed discussion of these networks except to point out that maximum pressure policies are also throughput optimal for these systems.

9.1. Networks with Alternate Routing

Consider the queueing network depicted in Figure 4. It has three servers (represented by circles) processing jobs in three buffers (represented by open rectangles). Server i processes jobs exclusively from buffer i, $i = 1, 2, 3$. There are four exogenous Poisson arrival processes that are independent. For $i = 1, 2, 3$, jobs from arrival process i go to buffer i to be processed by server i. Jobs from Arrival Process 4 can either go to Buffer 1 or 2. They are called discretionary jobs. The arrival rate for each process is given in the figure. The processing times for jobs in Buffers 2 and 3 are i.i.d., having exponential distribution with mean 1. The processing times for jobs in Buffer 1 are i.i.d. having a general distribution with mean 1. Jobs processed by Server 1 go to Buffer 3, and jobs processed by Servers 2 and 3 leave the network after processing.

When each server employs a nonidling service policy and processes jobs in FIFO order, the only decisions that are left for the network are the routing decisions for the discretionary jobs. This network has been studied by Dai and Kim (2003) to show that its stability region depends on the processing time distribution for Server 1. Assume that the join-shortest-queue routing policy is employed for the discretionary jobs, and a fair coin is used to break ties. They proved that when the processing time distribution for Server 1 is exponential, the network is unstable in the sense that the three-dimensional buffer-level process is not positive recurrent. They further demonstrated through simulation that the buffer level in Buffer 3 grows to infinity linearly as time goes to infinity. When the processing time distribution for Server 1 is hyperexponential with certain parameters, the network is stable in the sense that a certain Markov chain describing the network is positive recurrent.

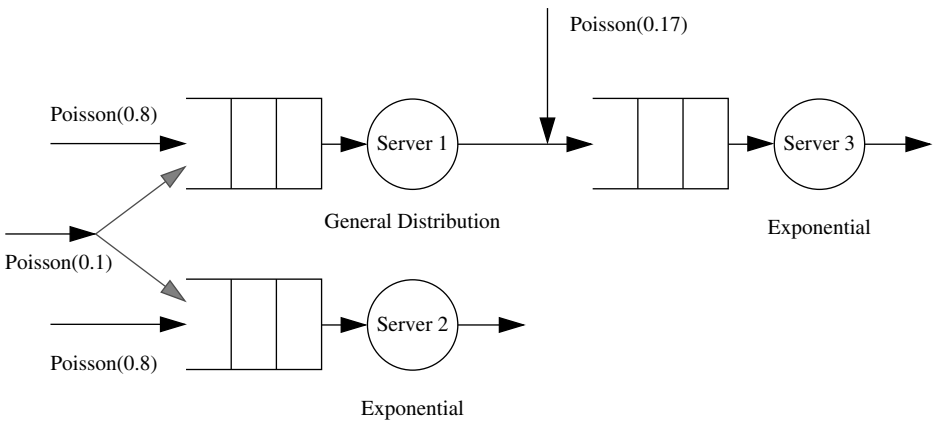
Their instability result is not surprising. When processing time distribution for Server 1 is exponential, Servers 1 and 2 are homogeneous. Thus, under the join-shortest-queue routing policy, half of the discretionary jobs go to Buffer 1, and these jobs will eventually go to Buffer 3. Thus, the traffic intensity of Server 3 is equal to

$$0.8 + 0.17 + 0.05 = 1.02 > 1,$$

causing the instability of the network. When processing time distribution for Server 1 is hyperexponential, the processing times have more variability, producing a larger queue in Buffer 1 than the one in Buffer 2, and hence fewer discretionary jobs joining Buffer 1. By choosing a parameter such that the hyperexponential distribution has high enough variability, the network is actually stable. Having network stability to depend on its processing time distributions is not attractive in practice. For the particular set of network parameters, it can be proven that a round-robin routing policy that routes 90% jobs to Buffer 2 stabilizes the network. Of course, as the arrival rates change, the percentage in the round-robin policy needs to be adjusted as well. Having the policy to depend on the arrival rates is not attractive either when the arrival rates are difficult to be reliably estimated.

We now describe our maximum pressure policies for the network. In turning the queueing network with alternate routes into our stochastic processing network framework, we need to introduce four input processors, one for each arrival process. The processing times are equal to interarrival times. Input Processor 4 is associated with two input activities, denoted as $(4, 1)$ and $(4, 2)$. Each time the processor completes an activity $(4, i)$ processing, a job goes to buffer i, $i = 1, 2$. The two input activities exemplify an extension of the stochastic processing networks in Harrison (2000) to allow external inputs with

Figure 4. A queueing network with alternate routes.



routing capabilities. The resulting stochastic processing network is unitary. The maximum pressure policy in Definition 1 amounts to the following: Discretionary jobs join the shorter queue among Buffers 1 and 2 (with an arbitrary tie-breaking rule); Servers 2 and 3 employ the nonidling service policy; Server 1 stops processing whenever

$$Z_3(t) > Z_1(t). \tag{31}$$

By forcing Server 1 to idle when the downstream buffer has more jobs than its buffer, the network is able to propagate its delay to the source nodes, making join–shortest-queue routing policy stable. This example hints at the difficulty in finding a pure local policy that is throughput optimal. If, in addition to throughput, other performance measures like average delay are important, one can consider a parameterized family of maximum pressure policies as in Corollary 1. In this case, Condition (31) is replaced by $\gamma_3 Z_3(t) > \gamma_1 Z_1(t) + \theta$ for some positive numbers γ_1, γ_3 and real number θ. For any fixed choice of parameters, the maximum pressure policy is throughput optimal. One can choose parameters to minimize average delay among maximum pressure policies.

Any maximum pressure policy is throughput optimal not only for the network depicted in Figure 4, but also for general multiclass queueing networks with alternative routes, a class of networks studied in Laws (1992). Figure 5 depicts such a network, which was first studied by Laws and Louth (1990) and later by Kelly and Laws (1993). There are two types of jobs, horizontal and vertical. For each type of job, there are two arrival activities. Each activity corresponds to a selection of a route that jobs will take. There are four service processors represented by circles. Each service processor can process jobs from one of the two constituency buffers, which are represented by open rectangular boxes. This network can be modeled by a stochastic processing network with a total of eight service activities and four input activities. The example in Figure 5 does not have internal routing decisions or probabilistic routing or feedback. A general multiclass queueing network with alternate routes can have all these features. Any nonpreemptive, nonprocessor-splitting maximum pressure policy is throughput optimal in such a network.

Multiclass queueing networks without routing capabilities were first introduced by Harrison (1988). They have been used to model re-entrant flows in semiconductor wafer fabrication facilities (Chen et al. 1988, Wein 1988, Connors et al. 1994, Lu et al. 1994). Multiclass queueing networks with alternate routes have the potential to model Internet Border Gateway routing (Basu et al. 2001, Gao and Rexford 2001, Labovitz et al. 2001). They also have the prospect to serve as road traffic models that support time-based or congestion-based pricing schemes (Fawcett and Robinson 2000, van Vuren and Smart 1990).

9.2. Networks of Data Switches

The Internet is supported by numerous interconnected data switches. A session of data packets from a source to a destination typically traverses a number of switches. The speed at which these switches process the packets influences the propagation delay of the Internet session. Data switches based on an input-queued crossbar architecture are attractive for use in high-speed networks (Dai and Prabhakar 2000; McKeown 1995, 1999; McKeown et al. 1999). Figure 6 depicts a network of three input-queued switches. Each switch has two input ports and two output ports.

When a session of packets enters the network, it contains information regarding how the packets traverse the network. We assume that these sessions are perpetual (relative to the time window of interest). Such a session is referred to as a *flow* in the rest of this section. Each flow corresponds to a sequence of input and output ports that the packets will traverse. (Each pair of input and output ports is traversed at most once by each flow.) At each input port,

Figure 5. A queueing network with alternate routes.

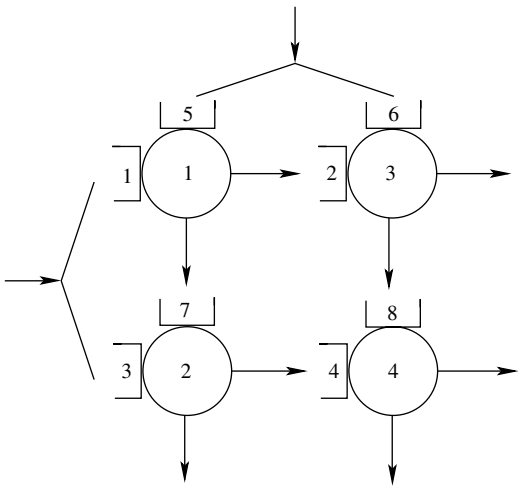
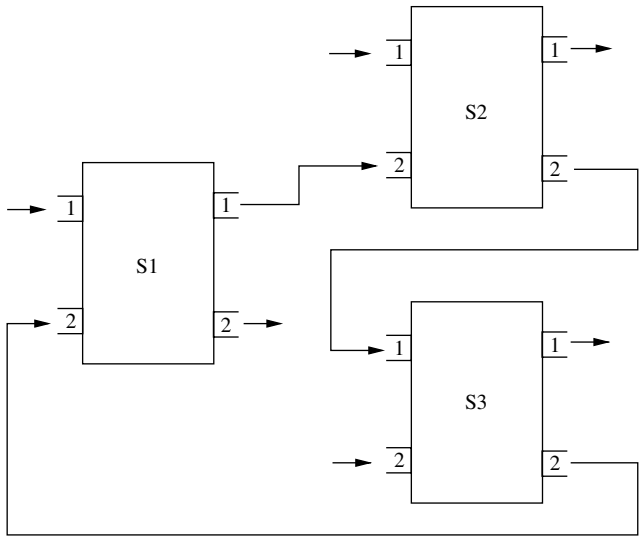


Figure 6. A network of input-queued switches.



we assume that there is an infinite-capacity buffer that can hold packets to be transmitted. We assume per-flow queueing. That is, each flow maintains a (logical) FIFO queue, called *flow queue*, at each input port. Each flow queue is associated with a pair of input and output ports at a switch. All packets in a flow queue are transmitted to the same output port. On the other hand, a pair of input and output ports may be associated with several flow queues if several flows traverse that input-output pair. A flow queue from flow c at input port ℓ_1 to be transmitted to output port ℓ_2 is denoted by (c, ℓ_1, ℓ_2). Both the input and output ports should be available to transmit a packet in the flow queue. At a switch, in each time slot, at most one packet is sent from each input port and at most one packet is sent to each output port. Although different switches may employ different time units for a time slot, we assume that each packet takes exactly one unit of time to transmit from a flow queue at the input port to the output port, and then instantaneously to the next flow queue at a downstream input port. The restrictive assumption is mainly for the ease of exposition. The conclusion in this section holds for the general case.

At the beginning of each time slot, each switch needs to find a matching of input and output ports that can simultaneously be active during the slot. It also needs to decide which flow queue to serve at each input port. Define the incidence matrix

$$H_{c,(\ell_1,\ell_2)} = \begin{cases} 1 & \text{if } \ell_1 \text{ and } \ell_2 \text{ are an input-output port pair at a switch and flow } c \text{ traverses the pair,} \\ 0 & \text{otherwise.} \end{cases} \tag{32}$$

Assume that α_c is the arrival rate of packets from flow c. It is intuitively clear that for the network of switches to be stabilizable it is necessary that

$$\sum_{c,\ell_2} \alpha_c H_{c,(\ell_1,\ell_2)} \leqslant 1 \quad \text{for each input port } \ell_1, \tag{33}$$

$$\sum_{c,\ell_1} \alpha_c H_{c,(\ell_1,\ell_2)} \leqslant 1 \quad \text{for each output port } \ell_2. \tag{34}$$

Any scheduling policy that stabilizes the network whenever (33) and (34) are satisfied is said to be throughput optimal.

For a single switch in isolation, it has been shown that certain policies that are based on maximum weight matchings between input and output ports are throughput optimal, where the weight for an input-output pair is based either on the queue size or the delay (Dai and Prabhakar 2000, McKeown et al. 1999, Tassiulas and Ephremides 1992). However, Andrews and Zhang (2001) provided a network of eight switches for which the maximum weight policy at each switch is not throughput optimal for the *network*. They further showed that the longest-in-network policy is throughput optimal (Andrews and Zhang 2001). A family of Birkhoff-von-Neumann-based policies were shown to achieve maximum throughput for networks of switches (Marsan et al. 2003). These policies use nonlocal information like arrival rates that may be difficult to estimate in some situations. In the remainder of this section, we are going to show that non-processor-splitting, nonpreemptive maximum pressure policies are throughput optimal for networks of switches. The maximum pressure policies are not purely local in that each switch makes scheduling decisions based on the flow queue lengths at the switch and their immediate downstream flow queues.

To state and prove our theorem, we use a stochastic processing network to model the network of input-queued switches. Flow queues serve as buffers in the stochastic processing. There are a total of $\mathbf{L}+1$ buffers, $\mathbf{L}$ internal buffers for flow queues plus the external Buffer 0 modeling the outside world. These internal buffers are indexed by $i = (c, \ell_1, \ell_2)$ representing the flow queue from flow c at input port ℓ_1 destined for output ℓ_2. Each flow c has an input processor to generate packets from Buffer 0 into the network at rate $\alpha_c = 1/m_j$, where j represents the input activity. Each input port is a service processor, and each output is a service processor. Each service activity requires two processors, a pair of input and output ports, to transmit a packet. Service activities are indexed by $j = (c, \ell_1, \ell_2)$. If service activity $j = (c, \ell_1, \ell_2)$ is taken, then a packet of flow c is transmitted from input port l_1 to output port l_2 at the end of a service completion. When activity $j = (c, \ell_1, \ell_2)$ is active, both input port (processor) ℓ_1 and output port (processor) ℓ_2 are occupied, preventing the transmission of packets from other flow queues through either port ℓ_1 or port ℓ_2. The total number of service activities is identical to the total number of internal buffers. Clearly, $m_j = 1$ for each service activity j.

THEOREM 10. *Assume that strong-law-of-large-numbers Assumption* (3) *is satisfied for each input activity j. Assume further that traffic conditions* (33) *and* (34) *are satisfied. The network of switches operating under any nonpreemptive, non-processor-splitting maximum pressure policy is stable.*

PROOF. We first verify that traffic conditions (33) and (34) are equivalent to the existence of a feasible solution (x, ρ) to the static planning problem (10)–(13) with $\rho \leqslant 1$. To see this, one can verify that the input-output matrix R in (5) can be written as

$$R_{i,j} = \begin{cases} -\alpha_c & \text{if input activity } j \text{ generates packets to buffer } i = (c, \ell_1, \ell_2), \\ 1 & \text{if service activity } j \text{ processes packets in buffer } i, \\ -1 & \text{if service activity } j \text{ processes packets that go next to buffer } i. \end{cases} \tag{35}$$

Define an $\mathbf{L} \times \mathbf{L}$ matrix $\hat{R}$ via

$$\hat{R}_{ij} = R_{ij} \quad \text{for each buffer } i \text{ and each service activity } j.$$

Then, we have $\hat{R} = (I - \hat{P})'$, where $\hat{P}_{i,i'} = 1$ if packets in flow queue i go next to flow queue i', and 0 otherwise.

Let (x, ρ) be a feasible solution to (10)–(13). Condition (12) implies that $x_j = 1$ for each input activity j. Let $\hat{x}$ be the remaining components of x, namely, $\hat{x}_j = x_j$ for each service activity. Condition (10) is equivalent to

$$\hat{R}\hat{x} = \lambda,$$

where, for each buffer $i = (c, \ell_1, \ell_2)$, $\lambda_i = \alpha_c$ if flow queue i is the first queue for flow c and $\lambda_i = 0$ otherwise. Because $\hat{R}$ is invertible, we have

$$\hat{x} = \hat{R}^{-1}\lambda.$$

For buffers i and i', $\hat{R}^{-1}_{i,i'}$ is the number of times that a packet in buffer i' will visit buffer i before it exits the network. Therefore, $\hat{R}^{-1}_{i,i'} = 1$ if $i = i'$ or i is a downstream flow queue of i', and 0 otherwise. One can verify that $\hat{x}_{(c,\ell_1,\ell_2)} = \alpha_c$ if flow c traverses a pair (ℓ_1, ℓ_2), and 0 otherwise.

For each input port ℓ_1, $A_{\ell_1,j} = 1$ if $j = (c, \ell_1, \ell_2)$ is a service activity at the flow queue j, and 0 otherwise. Similarly, for each output port ℓ_2, $A_{\ell_2,j} = 1$ if $j = (c, \ell_1, \ell_2)$ is a service activity at the flow queue j, and 0 otherwise. Thus, Condition (11) can be written as

$$\sum_{\text{service activities } j} A_{\ell_1,j}\hat{x}_j = \sum_{c,\ell_2} \hat{x}_{(c,\ell_1,\ell_2)} \leqslant \rho \quad \text{for each input port } \ell_1, \tag{36}$$

$$\sum_{\text{service activities } j} A_{\ell_2,j}\hat{x}_j = \sum_{c,\ell_1} \hat{x}_{(c,\ell_1,\ell_2)} \leqslant \rho \quad \text{for each output port } \ell_2, \tag{37}$$

and setting $\rho = 1$ results in the usual traffic intensity conditions (33) and (34). This proves the equivalence.

For the network of input-queued switches, because each activity is associated with exactly one buffer, it is strict Leontief. Thus, Assumption 1 is satisfied. One can further verify that $\mathscr{C} = \mathscr{N}$, and hence Condition (24)–(27) is equivalent to (10)–(13). Therefore, by Theorem 8, any nonprocessor-splitting maximum pressure policy that allows preemption achieves the optimal throughput. Because the switches operate in time slots, the configurations of the switches can be changed every time slot. As a consequence, the left-hand side of equation (58) in Appendix A.3 is bounded by 1, which implies Lemma 4, and hence, Theorems 2 and 8 still hold. Therefore, the nonpreemptive, nonprocessor-splitting maximum pressure policy is throughput optimal for the network of switches. □

Appendix A. Processing Network Equations and Fluid Limits

In this section, we first introduce network equations that are satisfied by a stochastic processing network operating under a general service policy. We then introduce fluid limits that connect a stochastic processing network with the corresponding fluid model introduced in §5. As a consequence, we show that the stability of a fluid model implies the stability of the corresponding stochastic processing network (Theorem 3).

A.1. Stochastic Processing Network Equations

For each activity j, we first define the counting process $S_j = \{S_j(t),\ t \geqslant 0\}$ associated with the processing requirement sequence $\eta_j = \{\eta_j(\ell),\ \ell \geqslant 0\}$. For each $t \geqslant 0$,

$$S_j(t) = \max\left\{n\colon \sum_{\ell=1}^{n} \eta_j(\ell) \leqslant t\right\}. \tag{38}$$

Because the service policy is assumed to be head-of-line, $S_j(t)$ is the number of activity j processing completions in t units of activity j processing time. Note that a unit of *activity j processing time* is not the same as a unit of *activity j busy time*. When the activity is employed at level a_j, one unit of activity j busy time is equal to a_j units of activity j processing time. We use $T_j(t)$ to denote the cumulative activity j processing time in $[0, t]$. Denoting $\mathbb{X}(t) = (Z(t), T(t))$ for $t \geqslant 0$, we call $\mathbb{X} = \{\mathbb{X}(t)\colon t \geqslant 0\}$ the stochastic processing network process.

Now we can write down the equations describing the dynamics of the stochastic processing network:

$$Z_i(t) = Z_i(0) + \sum_{i' \in \mathcal{I} \cup \{0\}} \sum_{j \in \mathcal{J}} \Phi^j_{i'i}(S_j(T_j(t))B_{ji'}) - \sum_{j \in \mathcal{J}} S_j(T_j(t))B_{ji} \quad \text{for each } t \geqslant 0 \text{ and } i \in \mathcal{I}, \tag{39}$$

$$Z_i(t) \geqslant 0 \quad \text{for each } t \geqslant 0 \text{ and } i \in \mathcal{I}, \tag{40}$$

$$\sum_{j \in \mathcal{J}} A_{kj}(T_j(t) - T_j(s)) = t - s \quad \text{for each } 0 \leqslant s \leqslant t \text{ and each input processor } k, \tag{41}$$

$$\sum_{j \in \mathcal{J}} A_{kj}(T_j(t) - T_j(s)) \leqslant t - s \quad \text{for each } 0 \leqslant s \leqslant t \text{ and each processor } k, \tag{42}$$

$$T \text{ is nondecreasing and } T(0) = 0. \tag{43}$$

Because quantity $T_j(t)$ is the cumulative amount of activity j processing time in $[0, t]$, $S_j(T_j(t))$ is the number of activity j processings completed by time t, and $\sum_{j \in \mathcal{J}} S_j(T_j(t))B_{ji'}$ is the total number of jobs that depart from buffer $i' \in \mathcal{I} \cup \{0\}$ in $[0, t]$. A fraction of that number, $\Phi^j_{i'i}(S_j(T_j(t))B_{ji'})$, goes to buffer i. Equation (39) is the flow balance equation. It says that the number of jobs in buffer i at time t equals the initial number plus the number of arrivals subtracted from the number of departures. Inequality (42) holds because each processor can spend at most $t - s$ units of time processing various activities from time s to t, and equation (41) holds because each input processor is assumed to never idle.

We note that equations (39)–(43) hold under any head-of-line service policy. They are called *stochastic processing network equations*. Under a specific service policy like a maximum pressure policy, there are additional equations for the network processes. These equations will be developed in §A.3.

A.2. Fluid Limits

Recall that $\mathbb{X} = (Z, T)$ is the stochastic network process describing the stochastic processing network, where $Z_i(t)$ is the number of jobs at time t in buffer i, and $T_j(t)$ is the cumulative activity j processing time in $[0, t]$. Clearly, $\mathbb{X}$ depends on a realization of sample path ω. We use $\mathbb{X}(\cdot, \omega)$ to denote the trajectory of the network process along sample path ω.

For each $r > 0$ and ω, define the scaled process $\mathbb{X}^r$ via

$$\bar{\mathbb{X}}^r(t, \omega) = r^{-1}\mathbb{X}(rt, \omega) \quad \text{for each } t \geqslant 0.$$

For a positive integer d, let $\mathbb{D}^d$ be the set of functions from $\mathbb{R}_+$ to $\mathbb{R}^d$ that are right continuous on $[0, \infty)$ and have left limits in $(0, \infty)$. Let $|\cdot|$ denote any norm in the Euclidean space $\mathbb{R}^d$. A sequence of functions $f_n \in \mathbb{D}^d$ is said to converge to an $f \in \mathbb{D}^d$ uniformly on compact (u.o.c.) sets, denoted as $f_n(\cdot) \to f(\cdot)$, if for each $t \geqslant 0$,

$$\lim_{n\to\infty} \sup_{0\leqslant s\leqslant t} |f_n(s) - f(s)| = 0.$$

Definition 6. A function $\bar{\mathbb{X}} = (\bar{Z}, \bar{T})$ is said to be a *fluid limit* of the processing network if there exists a sequence $r \to \infty$ and a sample path ω satisfying (3)–(4) such that

$$\lim_{r\to\infty} \bar{\mathbb{X}}^r(\cdot, \omega) \to \bar{\mathbb{X}}(\cdot).$$

To see the existence of a fluid limit, note that for each activity j and each sample path ω, for all $r > 0$,

$$\bar{T}_j^r(t, \omega) - \bar{T}_j^r(s, \omega) \leqslant t - s \quad \text{for } 0 \leqslant s \leqslant t.$$

Thus, the family of functions $\bar{T}_j^r(\cdot, \omega)$, $r > 0$ is equicontinuous; see, for example, Royden (1988). Thus, for each j there is a subsequence $r_n \to \infty$ as $n \to \infty$ such that

$$\lim_{n\to\infty} \bar{T}_j^{r_n}(\cdot, \omega) \to \bar{T}_j(\cdot)$$

for some continuous function $\bar{T}_j(\cdot)$. By a standard argument, one can find a further subsequence, still denoted by $\{r_n\}$ for notational convenience, such that

$$\lim_{n\to\infty} \bar{T}_j^{r_n}(\cdot, \omega) = \bar{T}_j(\cdot) \tag{44}$$

for each activity j.

To show that $\bar{Z}^{r_n}(\cdot, \omega)$ converges, we are going to use flow balance equation (39). Let us first focus on the last term in the right side of (39). By (3), for the fixed sample path ω,

$$\lim_{r\to\infty} S_j(rt, \omega)/r = \mu_j t \quad \text{for each } t \geqslant 0. \tag{45}$$

By (44) and (45), for each fixed t,

$$\begin{aligned}\lim_{n\to\infty} &\sum_j S_j(T_j(r_n t), \omega) B_{ji}/r_n \\ &= \sum_j B_{ji} \lim_{n\to\infty} \frac{S_j(T_j(r_n t), \omega)}{T_j(r_n t)} \frac{T_j(r_n t)}{r_n} \\ &= \sum_{j\in\mathcal{J}} B_{ji}\mu_j \bar{T}_j(t). \end{aligned} \tag{46}$$

Because for each fixed n, $\sum_j S_j(T_j(r_n t), \omega)B_{ji}/r_n$ is a nondecreasing function of t, and $\sum_{j\in\mathcal{J}} B_{ji}\mu_j \bar{T}_j(t)$ is a continuous function of t, convergence in (46) actually holds uniformly on compact sets; see, for example, Lemma 4.1 of Dai (1995).

Similarly, by using (44), (45), and (4), we have

$$\lim_{n\to\infty} \sum_{i'} \sum_j \Phi_{i'i}^j(S_j(T_j(r_n t))B_{ji'})/r_n = \sum_j \sum_{i'} \bar{T}_j(t)\mu_j B_{ji'} P_{i'i}^j. \tag{47}$$

It follows from (46), (47), and (39) that $\bar{Z}^{r_n}(\cdot) \to \bar{Z}(\cdot)$ with $\bar{Z}(0) = 0$ and $\bar{Z}$ satisfying (14). Clearly, $\bar{\mathbb{X}} = (\bar{Z}, \bar{T})$ also satisfies fluid model equations (15)–(18). Thus, the fluid limit $\bar{\mathbb{X}}$ is a fluid model solution to fluid model equations (14)–(18). We end this section by proving Theorem 3, which provides a connection between the stability of a stochastic processing network and the stability of the corresponding fluid model.

Proof of Theorem 3. Let ω be a sample path that satisfies (3) and (4). Let $\{r_n\}$ be a sequence such that $r_n \to \infty$ as $n \to \infty$. Consider the scaled sequence $\{\bar{\mathbb{X}}^{r_n}(t, \omega) = r_n^{-1}\mathbb{X}(r_n t, \omega),\ t \geqslant 0\colon\ n \geqslant 1\}$. By the arguments in the preceding three paragraphs, fluid limits exist. Namely, there exists a subsequence $\{r_{n'}\}$ such that

$$\bar{\mathbb{X}}^{r_{n'}}(\cdot, \omega) \to \bar{\mathbb{X}}(\cdot).$$

Because $\bar{\mathbb{X}} = (\bar{Z}, \bar{T})$ is also a fluid model solution with $\bar{Z}(0) = 0$, by the weak stability of the fluid model, $\bar{Z}(1) = 0$. Namely,

$$\lim_{n'\to\infty} \frac{Z(r_{n'}, \omega)}{r_{n'}} = 0. \tag{48}$$

Because $\{r_n\}$ is an arbitrary sequence with $r_n \to \infty$, (48) implies that

$$\lim_{t\to\infty} \frac{Z(t, \omega)}{t} = 0.$$

Thus, the stochastic processing network is pathwise stable. □

A.3. Fluid Limits Under a Maximum Pressure Policy

The main purpose of this section is to prove the following lemma, justifying that the fluid model under a maximum pressure policy is well defined by fluid model equations (14)–(17) and (20).

Lemma 4. *Consider a stochastic processing network operating under a preemptive, processor-splitting maximum pressure policy. Assume that the network satisfies Assumption* 1. *Each fluid limit satisfies fluid model equation* (20).

The proof of the lemma will be presented at the end of this section. For that, let $T^a(t)$ be the cumulative amount of time that allocation a has been employed in $[0, t]$. Under a maximum pressure policy, only extreme allocations are used. Thus,

$$T_j(t) = \sum_{a \in \mathscr{E}} a_j T^a(t) \quad \text{for each } t \geqslant 0 \text{ and } j \in \mathscr{J}. \tag{49}$$

Clearly,

$$T^a(\cdot) \text{ is nondecreasing} \quad \text{for each allocation } a \in \mathscr{E}, \tag{50}$$

$$\sum_{a \in \mathscr{E}} T^a(t) = t \quad \text{for each } t \geqslant 0. \tag{51}$$

Because each extreme allocation is in $\mathscr{A}$, one can check that (49)–(51) imply (41) and (42).

Under a maximum pressure policy, we modify the definition of the stochastic processing network process via

$$\mathbb{X}(t) = \big(Z(t), T^a(t), T_j(t)\colon\ a \in \mathscr{E},\ j \in \mathscr{J}\big).$$

Each $\mathbb{X}$ satisfies the stochastic processing network equations (39), (40), and (49)–(51). The fluid limits of $\mathbb{X}$ are defined analogously as in §A.2.

Lemma 5. *Assume that the EAA Assumption holds. Each fluid limit* $\overline{\mathbb{X}} = (\bar{Z}, \bar{T}^a, \bar{T}_j\colon\ a \in \mathscr{E},\ j \in \mathscr{J})$ *under a preemptive, processor-splitting maximum pressure policy satisfies fluid model equations* (14)–(17), *and the following equations*:

$$\bar{T}_j(t) = \sum_{a \in \mathscr{E}} a_j \bar{T}^a(t) \quad \textit{for each } t \geqslant 0 \textit{ and } j \in \mathscr{J}, \tag{52}$$

$$\bar{T}^a(\cdot) \textit{ is nondecreasing} \quad \textit{for each allocation } a \in \mathscr{E}, \tag{53}$$

$$\sum_{a \in \mathscr{E}} \bar{T}^a(t) = t \quad \textit{for each } t \geqslant 0, \tag{54}$$

$$\dot{\bar{T}}^a(t) = 0 \quad \textit{if } p\big(a, \bar{Z}(t)\big) < \max_{a' \in \mathscr{E}} p\big(a', \bar{Z}(t)\big). \tag{55}$$

Proof. Let $\overline{\mathbb{X}}$ be a fluid limit. Clearly, it satisfies (52)–(54). It remains to prove that $\overline{\mathbb{X}}$ satisfies (55).

Recall that $p(a, \bar{Z}(t)) = \bar{Z}(t) \cdot Ra$ is the network pressure under allocation a when the fluid level is $\bar{Z}(t)$. Suppose that $a \in \mathscr{E}$ and $p(a, \bar{Z}(t)) < \max_{a' \in \mathscr{E}} p(a', \bar{Z}(t))$. From Assumption 1, we can choose an $a^* \in \mathscr{E}$ such that

$$p\big(a^*, \bar{Z}(t)\big) = \max_{a' \in \mathscr{E}} p\big(a', \bar{Z}(t)\big)$$

and the fluid level $\bar{Z}_i(t) > 0$ for each constituent buffer i of a^*. Denote by $\mathscr{I}(a^*)$ the set of constituent buffers. Namely,

$$\mathscr{I}(a^*) = \left\{ i\colon\ \sum_j a_j^* B_{ji} > 0 \right\}.$$

Then, $\bar{Z}_i(t) > 0$ for all $i \in \mathscr{I}(a^*)$. Because $p(a, \bar{Z}(t)) < p(a^*, \bar{Z}(t))$ and $\min_{i \in \mathscr{I}(a^*)} \bar{Z}_i(t) > 0$, by the continuity of $\overline{\mathbb{X}}(\cdot)$, there exist $\epsilon > 0$ and $\delta > 0$ such that for each $\tau \in [t - \epsilon, t + \epsilon]$ and $i \in \mathscr{I}(a^*)$,

$$p\big(a, \bar{Z}(\tau)\big) + \delta \leqslant p\big(a^*, \bar{Z}(\tau)\big) \quad \text{and} \quad \bar{Z}_i(\tau) \geqslant \delta.$$

Thus, when n is sufficiently large, $p(a, Z(n\tau)) + n\delta/2 \leqslant p(a^*, Z(n\tau))$ and $Z_i(n\tau) \geqslant n\delta/2$ for each $i \in \mathscr{I}(a^*)$ and each $\tau \in [t - \epsilon, t + \epsilon]$. Choosing $n > 2\mathbf{J}/\delta$, then for each $\tau \in [n(t - \epsilon), n(t + \epsilon)]$, we have

$$p(a, Z(\tau)) < p(a^*, Z(\tau)), \tag{56}$$

$$Z_i(\tau) \geqslant \mathbf{J} \quad \text{for each } i \in \mathscr{I}(a^*). \tag{57}$$

Condition (57) implies that a^* is a feasible allocation at any time $\tau \in [n(t - \epsilon), n(t + \epsilon)]$; i.e., $a^* \in \mathscr{E}(\tau)$. Following (56) and the definition of a (preemptive-resume) maximum pressure policy, the allocation a will not be employed during time interval $[n(t - \epsilon), n(t + \epsilon)]$. Therefore,

$$T^a(n(t + \epsilon)) - T^a(n(t - \epsilon)) = 0, \tag{58}$$

which implies $\bar{T}^a(t + \epsilon) - \bar{T}^a(t - \epsilon) = 0$, and hence $\dot{\bar{T}}_a(t) = 0$. □

Proof of Lemma 4. Let $\overline{\mathbb{X}}$ be a fluid limit. We would like to prove that $(\bar{Z}, \bar{T})$ satisfies fluid model equation (20). By Lemma 5 and the fact that $\sum_{a \in \mathscr{E}} \dot{\bar{T}}^a(t) = 1$,

$$\sum_{a \in \mathscr{E}} \dot{\bar{T}}_a(t) p\big(a, \bar{Z}(t)\big) \geqslant p\big(a', \bar{Z}(t)\big) \quad \text{for all } a' \in \mathscr{E}. \tag{59}$$

Now,

$$\begin{aligned} R\dot{\bar{T}}(t) \cdot \bar{Z}(t) &= \sum_{i \in \mathscr{J}} \bar{Z}_i(t) \sum_{j \in \mathscr{J}} R_{ij} \dot{\bar{T}}_j(t) \\ &= \sum_{i \in \mathscr{J}} \bar{Z}_i(t) \sum_{j \in \mathscr{J}} R_{ij} \sum_{a' \in \mathscr{E}} a'_j \dot{\bar{T}}^{a'}(t) \\ &= \sum_{a' \in \mathscr{E}} \dot{\bar{T}}^{a'}(t) p\big(a', \bar{Z}(t)\big) \\ &\geqslant p\big(a, \bar{Z}(t)\big) = Ra \cdot \bar{Z}(t). \end{aligned}$$

The preceding inequality together with (19) proves (20). □

Appendix B. Proofs

In this section, we provide proofs for Theorems 1, 5, 7, 9, and a few lemmas.

Proof of Theorem 1. Assume that the stochastic processing network is pathwise stable under some service policy. Fix a sample path that satisfies (3)–(4) and (8). Let $(\bar{Z}, \bar{T})$ be a fluid limit of (Z, T) along the sample path. Following

the arguments in §A.2, such a limit exists and satisfies the fluid model equations (14)–(18). Because the stochastic processing network is stable, $\bar{Z}(t) = 0$ for $t \geqslant 0$. For each activity j, let $x_j = \dot{\bar{T}}_j(1)$. It is easy to see that $x = (x_j)$ satisfies (9)–(13) with $\rho = 1$. □

Proof of Theorem 5. Suppose that $(\tilde{x}, \rho)$ is a feasible solution to (9)–(12). From Assumption 2, there exists an $\hat{x} \geqslant 0$ such that $R\hat{x} > 0$. Because for each input activity j, $R_{ij} = -\mu_j B_{j0} P^j_{0i} \leqslant 0$, we can set $\hat{x}_j = 0$ for each input activity j so that $R\hat{x} > 0$ still holds. As a consequence, $\sum_j A_{kj}\hat{x}_j = 0$ for each input processor k. Clearly, $\hat{x}$ can be scaled so that $\sum_j A_{kj}\hat{x}_j \leqslant (1-\rho)$ for each service processor k. Let $x^* = \tilde{x} + \hat{x}$. One can check that $x^* \in \mathcal{A}$, and $Rx^* = R\tilde{x} + R\hat{x} = R\hat{x} \geqslant \delta e$, where $\delta = \min_i \sum_j R_{ij}\hat{x}_j > 0$. By (20),

$$R\dot{\bar{T}}(t) \cdot \bar{Z}(t) \geqslant Rx^* \cdot \bar{Z}(t) \geqslant \delta \sum_i \bar{Z}_i(t) \geqslant \delta \|\bar{Z}(t)\|,$$

where $\|\cdot\|$ is the Euclidean norm on $\mathbb{R}^{\mathbf{I}}$. Therefore,

$$\dot{f}(t) = 2\dot{\bar{Z}}(t) \cdot \bar{Z}(t) = -2R\dot{\bar{T}}(t) \cdot \bar{Z}(t) \leqslant -2\delta\|\bar{Z}(t)\| = -2\delta\sqrt{f(t)}.$$

It follows that $\bar{Z}(t) = 0$ for $t \geqslant \|\bar{Z}(0)\|/\delta$. □

Proof of Theorem 7. Because the proof is analogous to the proof of Theorem 1, we present an outline, highlighting the differences between the two proofs. As in §A.3, for each allocation $a \in \mathcal{N}$, let $T^a(t)$ be the cumulative amount of time that allocation a has been employed in $[0, t]$. Because non-processor-splitting is assumed, equations (49)–(51) in §A.3 hold, with $\mathcal{N}$ replacing $\mathcal{C}$. Again as in §A.3, under a non-processor-splitting service policy, we modify the definition of the stochastic processing network process via

$$\mathbb{X}(t) = \big(Z(t), T^a(t), T_j(t)\colon a \in \mathcal{N},\ j \in \mathcal{J}\big).$$

Each $\mathbb{X}$ satisfies the stochastic processing network equations (39), (40), and (49)–(51). The fluid limits of $\mathbb{X}$ are defined analogously as in §A.2. Assume that the stochastic processing network is pathwise stable under some non-processor-splitting service policy. Fix a sample path that satisfies (3)–(4) and (8). Let $(\bar{Z}, \bar{T}^a, \bar{T}_j\colon a \in \mathcal{N},\ j \in \mathcal{J})$ be a fluid limit of $\mathbb{X}$ along the sample path. Similar to the arguments in §A.2, such a limit exists and satisfies the fluid model equations (14)–(18), and (52)–(54) with $\mathcal{N}$ replacing $\mathcal{C}$. Because the stochastic processing network is stable, $\bar{Z}(t) = 0$ for $t \geqslant 0$. For each allocation $a \in \mathcal{N}$, let $\pi_a = \dot{\bar{T}}^a(1)$. It is easy to see that $\pi = (\pi_a)$ is a feasible solution that satisfies (24)–(27). □

Proof of Lemma 1. For each processor k, let $\mathcal{J}(k)$ be the set of possible activities that the processor can take. We make the convention that when a processor is idle, it takes on Activity 0. (Note that idling is not an activity as defined in §2.) Thus,

$$\mathcal{J}(k) = \begin{cases} \{j\colon A_{kj} = 1\} & \text{for input processor } k, \\ \{0\} \cup \{j\colon A_{kj} = 1\} & \text{for service processor } k. \end{cases} \tag{60}$$

We prove the lemma by contradiction. Suppose that there exists an allocation $a \in \mathcal{C}$ such that $0 < a_{\tilde{j}} < 1$ for some activity $\tilde{j} \in \mathcal{J}$. Let $\tilde{k}$ be the processor processing activity $\tilde{j}$. For each $j \in \mathcal{J}(\tilde{k})$, we define a new allocation b^j by modifying the allocation a in the following way. For activities $j' \notin \mathcal{J}(\tilde{k})$, we keep the activity level $a_{j'}$; processor $\tilde{k}$ employs activity j at a 100% level. Clearly, b^j is a feasible allocation for each $j \in \mathcal{J}(\tilde{k})$. It follows that $a = \sum_{j \in \mathcal{J}(\tilde{k})} a_j b^j$, where, for a service processor $\tilde{k}$, we set $a_0 = 1 - \sum_{j \in \mathcal{J}(\tilde{k}),\, j \neq 0} a_j$. Because $\sum_{j \in \mathcal{J}(\tilde{k})} a_j = 1$, $\tilde{j} \in \mathcal{J}(\tilde{k})$, and $a_{\tilde{j}} < 1$ by assumption, a is a proper linear combination of feasible allocations. Thus, a is not an extreme allocation, contradicting the assumption that a is an extreme allocation. Therefore, any extreme allocation must be an integer allocation. On the other hand, any feasible integer allocation must be extreme. Hence, $\mathcal{C} = \mathcal{N}$. □

To prove Theorem 9, we first establish a lemma.

Lemma 6. *Define $\hat{\eta}_j(t)$ to be the residual processing time for activity j at time t. Then, for any sample path ω satisfying* (3) *and* (29),

$$\lim_{t\to\infty} \hat{\eta}_j(t)/t = 0.$$

Proof. It is straightforward to show that

$$\hat{\eta}_j(t) \leqslant \max_{1 \leqslant \ell \leqslant S_j(t)+1} \eta_j(\ell),$$

where $S_j(t)$, as defined in (16), is the number of activity j processing completions in t units of activity j processing time. It follows that

$$\hat{\eta}_j^{1+\epsilon_j}(t) \leqslant \max_{1 \leqslant \ell \leqslant S_j(t)+1} \eta_j^{1+\epsilon_j}(\ell) \leqslant \sum_{1 \leqslant \ell \leqslant S_j(t)+1} \eta_j^{1+\epsilon_j}(\ell).$$

Because $S_j(t) \to \infty$ as $t \to \infty$, we have

$$\lim_{t\to\infty} \sum_{1 \leqslant \ell \leqslant S_j(t)+1} \eta_j^{1+\epsilon_j}(\ell)/(S_j(t)+1) = m'_j.$$

Therefore,

$$\limsup_{t\to\infty} \hat{\eta}_j^{1+\epsilon_j}(t)/t \leqslant \lim_{t\to\infty} \left(\sum_{1 \leqslant \ell \leqslant S_j(t)+1} \eta_j^{1+\epsilon_j}(\ell)/(S_j(t)+1) \right) \cdot (S_j(t)+1)/t = m'_j\mu_j,$$

from which we have

$$\lim_{t\to\infty} \hat{\eta}_j(t)/t = 0. \quad \square$$

PROOF OF THEOREM 9. Let ω be a sample path that satisfies (3)–(4) and (29). Let $(\bar{Z}, \bar{T})$ be a fluid limit along the sample path. We only need to show that fluid model equation (55) is satisfied. The proof is similar to the proof of Lemma 4.

First, by the definition of a maximum pressure policy, for any allocation $a \notin \mathscr{E}$, $T^a(t) = 0$ for $t \geqslant 0$. As a consequence, we have $\sum_{a\in\mathscr{E}} T^a(t) = t$ for $t \geqslant 0$. It follows that $\sum_{a\in\mathscr{E}} \bar{T}^a(t) = t$ for $t \geqslant 0$. Recall that, as in (6), $p(a, \bar{Z}(t)) = \bar{Z}(t) \cdot Ra$ is the network pressure under allocation a.

Suppose that $a \in \mathscr{E}$ and $p(a, \bar{Z}(t)) < \max_{a'\in\mathscr{E}} p(a', \bar{Z}(t))$. From Assumption 1, we can choose an $a^* \in \mathscr{E}$ such that

$$p\big(a^*, \bar{Z}(t)\big) = \max_{a'\in\mathscr{E}} p\big(a', \bar{Z}(t)\big)$$

and the fluid level $\bar{Z}_i(t) > 0$ for all buffers i with $\sum_j a^*_j B_{ji} > 0$. Following the proof of Lemma 3, one has that

$$p\big(j_{\hat{k}}(a), \bar{Z}(t)\big) < p\big(j_{\hat{k}}(a^*), \bar{Z}(t)\big)$$

for some processor $\hat{k}$. Let

$$\hat{j} = j_{\hat{k}}(a) \quad \text{and} \quad j^* = j_{\hat{k}}(a^*).$$

Now we construct allocation $\hat{a} = a - e_{\hat{j}} + e_{j^*}$. In other words, allocation $\hat{a}$ is exactly the same as allocation a except that processor $\hat{k}$ takes activity j^* instead of $\hat{j}$. Obviously, $\hat{a} \in \mathscr{E}$ and

$$\begin{aligned} p\big(\hat{a}, \bar{Z}(t)\big) &= p\big(a, \bar{Z}(t)\big) - p\big(\hat{j}, \bar{Z}(t)\big) + p\big(j^*, \bar{Z}(t)\big) \\ &> p\big(a, \bar{Z}(t)\big). \end{aligned}$$

We next show that there exists an $\epsilon > 0$ such that for any large number n and each $\tau \in [n(t-\epsilon), n(t+\epsilon)]$,

$$p(a, Z(\tau)) < p(\hat{a}, Z(\tau)), \tag{61}$$

$$a \in \mathscr{E}(\tau) \text{ implies that } \hat{a} \in \mathscr{E}(\tau);$$
$$\text{namely, } \hat{a} \text{ is a feasible allocation as long as } a \text{ is.} \tag{62}$$

We first assume that $j^* \neq 0$. Denote by $\mathscr{I}(j^*)$ the set of constituency buffers of activity j^* that have potential positive output flows. Namely,

$$\mathscr{I}(j^*) = \{i\colon B_{j^*i} > 0\}.$$

Then, $\bar{Z}_i(t) > 0$ for all $i \in \mathscr{I}(j^*)$. Because $p(\hat{a}, \bar{Z}(t)) > p(a, \bar{Z}(t))$ and $\min_{i\in\mathscr{I}(j^*)} \bar{Z}_i(t) > 0$, by the continuity of $\bar{\mathbb{X}}(\cdot)$, there exist $\epsilon > 0$ and $\delta > 0$ such that for each $\tau \in [t-\epsilon, t+\epsilon]$ and $i \in \mathscr{I}(j^*)$,

$$p\big(a, \bar{Z}(\tau)\big) + \delta \leqslant p\big(a^*, \bar{Z}(\tau)\big) \quad \text{and} \quad \bar{Z}_i(\tau) \geqslant \delta.$$

Thus, when n is sufficiently large, $p(a, Z(n\tau)) + n\delta/2 \leqslant p(\hat{a}, Z(n\tau))$ and $Z_i(n\tau) \geqslant n\delta/2$ for each $i \in \mathscr{I}(j^*)$ and each $\tau \in [t-\epsilon, t+\epsilon]$. Choosing $n > 2\mathbf{J}/\delta$, then for each $\tau \in [n(t-\epsilon), n(t+\epsilon)]$, we have

$$Z_i(\tau) \geqslant \mathbf{J} \quad \text{for each } i \in \mathscr{I}(j^*). \tag{63}$$

Condition (63) implies that (62) holds, thus proving (62) and (61). When $j^* = 0$, (62) clearly holds for any $\epsilon > 0$ and any n. The proof of (61) is identical to the case when $j^* \neq 0$.

Following the definition of the maximum pressure policy and (61), if allocation a is not employed at time $n(t-\epsilon)$, it will not be employed during the time interval $[n(t-\epsilon),\ n(t+\epsilon)]$. If allocation a is employed at time $n(t-\epsilon)$, it will not be deployed during time interval $[n(t-\epsilon) + \hat{\eta}(n(t-\epsilon)), n(t+\epsilon)]$, where $\hat{\eta}(n(t-\epsilon))$ is the longest residual processing time among activities j that are active under allocation a at time $n(t-\epsilon)$. In either case, we have

$$T^a(n(t+\epsilon)) - T^a(n(t-\epsilon)) \leqslant \max_{j\in\mathscr{J}} \hat{\eta}_j(n(t-\epsilon)). \tag{64}$$

It follows from Lemma 6 that

$$\lim_{n\to\infty} n^{-1}\big(T^a(n(t+\epsilon)) - T^a(n(t-\epsilon))\big) = 0.$$

Thus, $\bar{T}^a(t+\epsilon) - \bar{T}^a(t-\epsilon) = 0$, and hence $\dot{\bar{T}}^a(t) = 0$. □

Acknowledgments

The authors thank Sasha Stolyar for sending his preprints (Andrews et al. 2004, Stolyar 2004) to them. His papers and Harrison's recent papers (2000, 2002, 2003) on stochastic processing networks have inspired this research. They also thank Sasha Stolyar for bringing the paper of Tassiulas and Bhattacharya (2000) to their attention. They thank Ruth Williams for various discussions that sharpened their understanding of processor-splitting policies. Research was supported in part by National Science Foundation grant DMI-0300599, by a Chinese National Science Foundation grant, and by TLI-AP, a partnership between National University of Singapore and Georgia Institute of Technology.

References

Andradóttir, S., H. Ayhan, D. G. Down. 2003. Dynamic server allocation for queueing networks with flexible servers. *Oper. Res.* **51** 952–968.

Andrews, M., L. Zhang. 2001. Achieving stability in networks of input-queued switches. *Proc. IEEE INFOCOM* **3** 1673–1679, Anchorage, AK.

Andrews, M., K. Kumaran, K. Ramanan, A. Stolyar, R. Vijayakumar, P. Whiting. 2004. Scheduling in a queueing system with asynchronously varying service rates. *Probab. Engrg. Inform. Sci.* **18** 191–217.

Armony, M. 1999. Queueing networks with interacting service resources. Ph.D. thesis, Engineering-Economic Systems and Operations Research, Stanford University, Stanford, CA.

Armony, M., N. Bambos. 2003. Queueing dynamics and maximal throughput scheduling in switched processing systems. *Queueing Systems: Theory Appl.* **44**(3) 209–252.

Basu, A., C.-H. L. Ong, A. Rasala, F. B. Shepherd, G. Wilfong. 2001. Route oscillations in I-BGP. Technical report, Bell Labs, Lucent Technologies.

Bell, S. L., R. L. Williams. 2001. Dynamic scheduling of a system with two parallel servers in heavy traffic with resource pooling: Asymptotic optimality of a threshold policy. *Ann. Appl. Probab.* **11** 608–649.

Bell, S. L., R. J. Williams. 2004. Dynamic scheduling of a parallel server system in heavy traffic with complete resource pooling: Asymptotic optimality of a threshold policy. Preprint.

Bertsimas, D., I. C. Paschalidis, J. Tsitsiklis. 1994. Optimization of multiclass queueing networks: Polyhedral and nonlinear characterizations of achievable performance. *Ann. Appl. Probab.* **4** 43–75.

Bramson, M. 1994. Instability of FIFO queueing networks. *Ann. Appl. Probab.* **4** 414–431.

Bramson, M. 1998. State space collapse with application to heavy traffic limits for multiclass queueing networks. *Queueing Systems: Theory Appl.* **30** 89–148.

Bramson, M., R. J. Williams. 2003. Two workload properties for Brownian networks. *Queuing Systems* **45**(3) 191–221.

Chen, H., D. Yao. 1993. Dynamic scheduling of a multiclass fluid network. *Oper. Res.* **41** 1104–1115.

Chen, H., J. M. Harrison, A. Mandelbaum, A. van Ackere, L. M. Wein. 1988. Empirical evaluation of a queueing network model for semiconductor wafer fabrication. *Oper. Res.* **36** 202–215.

Connors, D., G. Feigin, D. Yao. 1994. Scheduling semiconductor lines using a fluid network model. *IEEE Trans. Robotics Automation* **10** 88–98.

Dai, J. G. 1995. On positive Harris recurrence of multiclass queueing networks: A unified approach via fluid limit models. *Ann. Appl. Probab.* **5** 49–77.

Dai, J. G. 1999. Stability of fluid and stochastic processing networks. MaPhySto Miscellanea Publication, No. 9, Denmark.

Dai, J. G., B. Kim. 2003. Stability of Jackson type networks with alternate routes. Preprint.

Dai, J. G., S. P. Meyn. 1995. Stability and convergence of moments for multiclass queueing networks via fluid limit models. *IEEE Trans. Automatic Control* **40** 1889–1904.

Dai, J. G., B. Prabhakar. 2000. The throughput of data switches with and without speedup. *Proc. IEEE INFOCOM*, Vol. 2. Tel Aviv, Israel, 556–564.

Dantzig, G. B. 1951. The programming of interdependent activities: Mathematical models. T. C. Koopmans, ed. *Activity Analysis of Production and Allocation*. John Wiley and Sons, New York, 19–32.

Fawcett, J., P. Robinson. 2000. Adaptive routing for road traffic. *IEEE Comput. Graphics Appl.* **20** 46–53.

Gans, N., G. van Ryzin. 1997. Optimal control of a multiclass, flexible queueing system. *Oper. Res.* **45** 677–693.

Gans, N., G. Koole, A. Mandelbaum. 2003. Telephone call centers: A tutorial and literature review. *Manufacturing Service Oper. Management* **5** 79–141.

Gao, L., J. Rexford. 2001. Stable Internet routing without global coordination. *IEEE/ACM Trans. Networking* **9** 681–692.

Harrison, J. M. 1988. Brownian models of queueing networks with heterogeneous customer populations. W. Fleming, P. L. Lions, eds. *Stochastic Differential Systems, Stochastic Control Theory and Their Applications*. *The IMA Volumes in Mathematics and Its Applications*, Vol. 10. Springer, New York, 147–186.

Harrison, J. M. 2000. Brownian models of open processing networks: Canonical representation of workload. *Ann. Appl. Probab.* **10** 75–103.

Harrison, J. M. 2002. Stochastic networks and activity analysis. Y. Suhov, ed. *Analytic Methods in Applied Probability*. In Memory of Fridrik Karpelevich. American Mathematical Society, Providence, RI.

Harrison, J. M. 2003. A broader view of Brownian networks. *Ann. Appl. Probab.* **13** 1119–1150.

Harrison, J. M., M. J. López. 1999. Heavy traffic resource pooling in parallel-server systems. *Queueing Systems: Theory Appl.* **33** 339–368.

Harrison, J. M., V. Nguyen. 1993. Brownian models of multiclass queueing networks: Current status and open problems. *Queueing Systems: Theory Appl.* **13** 5–40.

Kelly, F. P., C. N. Laws. 1993. Dynamic routing in open queueing networks: Brownian models, cut constraints and resource pooling. *Queueing Systems: Theory Appl.* **13** 47–86.

Koopmans, T. C., ed. 1951. *Activity Analysis of Production and Allocation*. John Wiley and Sons, New York.

Kumar, S., P. R. Kumar. 1994. Performance bounds for queueing networks and scheduling policies. *IEEE Trans. Automatic Control* **AC-39** 1600–1611.

Labovitz, C., A. Ahuja, R. Wattenhofer, S. Venkatachary. 2001. The impact of Internet policy and topology on delayed routing convergence. *Proc. IEEE INFOCOM*, Vol. 1. Anchorage, AK, 537–546.

Laws, C. N. 1992. Resource pooling in queueing networks with dynamic routing. *Adv. Appl. Probab.* **24** 699–726.

Laws, C. N., G. M. Louth. 1990. Dynamic scheduling of four-station queueing networks. *Probab. Engrg. Inform. Sci.* **4** 131–156.

Lu, S. C. H., D. Ramaswamy, P. R. Kumar. 1994. Efficient scheduling policies to reduce mean and variance of cycle-time in semiconductor manufacturing plants. *IEEE Trans. Semiconductor Manufacturing* **7** 374–385.

Marsan, M. A., P. Giaccone, E. Leonardi, F. Neri. 2003. On the stability of local scheduling policies in networks of packet switches with input queues. *J. Selected Areas Comm.* **21**(4) 642–655.

McKeown, N. 1995. Scheduling algorithms for input-queued cell switches. Ph.D. thesis, University of California, CA.

McKeown, N. 1999. iSLIP: A scheduling algorithm for input-queued switches. *IEEE Trans. Networking* **7** 188–201.

McKeown, N., A. Mekkittikul, V. Anantharam, J. Walrand. 1999. Achieving 100% throughput in an input-queued switch. *IEEE Trans. Comm.* **47** 1260–1267.

Meyn, S. 1997. Stability and optimization of queueing networks and their fluid models. *Mathematics of Stochastic Manufacturing Systems* (*Williamsburg, VA, 1996*). *Lectures in Applied Mathematics*, Vol. 33. American Mathematical Society, Providence, RI, 175–199.

Royden, H. L. 1988. *Real Analysis*, 3rd ed. Prentice Hall, New York.

Seidman, T. I. 1994. "First come, first served" can be unstable! *IEEE Trans. Automatic Control* **39** 2166–2171.

Squillante, M. S., C. H. Xia, D. D. Yao, L. Zhang. 2001. Threshold-based priority policies for parallel-server systems with affinity scheduling. *IEEE Amer. Control. Conf.*, 2992–2999.

Stolyar, A. L. 1995. On the stability of multiclass queueing networks: A relaxed sufficient condition via limiting fluid processes. *Markov Processes Related Fields* **1** 491–512.

Stolyar, A. L. 2004. MaxWeight scheduling in a generalized switch: State space collapse and workload minimization in heavy traffic. *Ann. Appl. Probab.* **14** 1–53.

Tassiulas, L. 1995. Adaptive back-pressure congestion control based on local information. *IEEE Trans. Automatic Control* **40** 236–250.

Tassiulas, L., P. B. Bhattacharya. 2000. Allocation of interdependent resources for maximal throughput. *Stochastic Models* **16** 27–48.

Tassiulas, L., A. Ephremides. 1992. Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks. *IEEE Trans. Automatic Control* **37**(12) 1936–1948.

Tassiulas, L., A. Ephremides. 1993. Dynamic server allocation to parallel queues with randomly varying connectivity. *IEEE Trans. Inform. Theory* **39** 466–478.

Thomas, L. J., J. O. McClain. 1993. An overview of production planning. S. C. Graves, A. H. G. R. Kan, P. Zipkin, eds. *Logistics of Production and Inventory. Handbooks in Operations Research and Management Science*, Vol. 4. North-Holland, Amsterdam, The Netherlands.

van Vuren, T., M. B. Smart. 1990. Route guidance and road pricing—Problems, practicalities and possibilities. *Transport Rev.* **10** 269–283.

Wein, L. M. 1988. Scheduling semiconductor wafer fabrication. *IEEE Trans. Semiconductor Manufacturing* **1** 115–130.

Wein, L. M. 1992. Scheduling networks of queues: Heavy traffic analysis of a multistation network with controllable inputs. *Oper. Res.* **40** S312–S334.

Williams, R. J. 1998. Diffusion approximations for open multiclass queueing networks: Sufficient conditions involving state space collapse. *Queueing Systems: Theory Appl.* **30** 27–88.

Williams, R. J. 2000. On dynamic scheduling of a parallel server system with complete resource pooling. D. R. McDonald, S. R. E. Tuners, eds. *Analysis of Communication Networks: Call Centres, Traffic and Performance. Fields Institute Communications*, Vol. 8. American Mathematical Society, Providence, RI, 49–71.