1

Introduction

This book considers a broad class of stochastic system models, focusing on questions and methods related to long-run “stability.” To be more precise, we consider stochastic models of multi-resource processing systems, assuming throughout that average input rates and average processing rates are time-invariant, and we focus on the following questions: Do the processing resources have enough capacity to handle the given or hypothesized load, and if so, how can the resources be deployed dynamically to ensure that statistical equilibrium is achieved over the long run?

The first four sections of this introductory chapter strive to explain the breadth and variety of models considered, beginning with the question of how to name them. Sections 1.5 and 1.6 elaborate on the notion of system stability. Section 1.7 describes the structure of the book and its intended audience, and Section 1.8 contains brief comments about sources and literature.

1.1 About the Title of This Book

Our focus is on models of man-made systems in which servers (that is, capacity-constrained processing resources) undertake various activities to satisfy the needs of externally generated jobs. Emphasizing the purpose for which such systems exist, we call these models *processing* networks. The “jobs” to be processed may be digital files requiring transfers, telephone calls from customers seeking information, manufacturing lots that require a particular factory operation, or other possibilities, depending on the application domain.

There is a large literature in applied mathematics concerned with the performance degradation caused by stochastic variability in the functioning of such systems. In that literature, mathematical models are used to explicate the perplexing phenomenon whereby some arriving jobs experience lengthy delays before their processing is completed, even though naive calculations based on average arrival rates and average processing rates indicate that resource capacities are adequate to handle the offered load. Such models are described throughout most of this book as *stochastic* processing networks (SPNs), but for brevity the modifier has been dropped in the book's title.

Queueing theory is a branch of applied mathematics that deals with performance degradation due to stochastic variability, and virtually all of the literature to which this book refers can be placed beneath that very broad umbrella. In fact, "queueing network" might seem to be a reasonable substitute for "stochastic processing network," but the former term has an established meaning in the literature of applied probability (see Sections 1.3, 1.4, and 2.6 for elaboration), and it is narrower than what we mean by the latter term. In particular, the term "queueing network" is generally understood to *not* allow simultaneous resource possession (as occurs in models of bandwidth sharing), nor to allow simultaneous input requirements (as occurs in models of assembly and matching), but those two features *are* allowed in our conception of a stochastic processing network.

What does the word "network" mean in our context? The non-mathematical definition of that word involves multiple distinct entities, such as power plants or supermarkets, that are connected in some way. Adapting that definition to our setting, one may safely say that all the models encountered in this book involve two or more distinct processing resources (or servers, or server pools) that are connected by a workflow. That is, all our models involve a network of capacity-constrained resources that differ in their capabilities and are connected by the need to collaborate, either sequentially or simultaneously, in processing jobs of different types. That high-level characterization is rather vague, and applied mathematicians often attach to the word "network" a narrower meaning that involves *arcs* and *nodes* in a graph structure. Indeed, many important stochastic network models, notably models of communication networks, do involve an underlying graph structure. Such models are somewhat special, however, and in other application domains, the arcs-and-nodes framework of classical network theory is overly restrictive.

1.2 Activity Analysis

Our conception of a stochastic processing network is derived from, and seeks to extend, the deterministic linear model of an operation or enterprise that was developed in the mid-twentieth century by mathematicians and economists working in the area that Koopmans (1951) called “activity analysis.” Today that term is viewed as an old-fashioned synonym for “linear programming” (LP) or “linear optimization,” but the emphasis here is not on optimization per se. Rather, we seek to develop stochastic analogs of the linear input–output models that define the “feasible region” in various classical LP problem formulations.

In the version of activity analysis that we shall adopt, an economic system or subsystem is characterized in terms of three basic elements: processing *resources* with fixed capacities per time unit; processing *activities* that may be undertaken using those resources; and units of flow, initially referred to as *materials*, that are created, destroyed, or modified by processing activities. In the standard treatment, activity levels and material quantities are treated as continuous variables, but the time parameter (in the case of dynamic models) is discrete. Thus one might say that classical activity analysis uses deterministic fluid models with a discrete time parameter.

The term “server” is commonly used as a synonym for “resource,” and we shall follow that practice. However, in many applications a “resource” consists of multiple distinct but functionally interchangeable entities, such as electronic testers, vertical turret lathes, or customer service representatives. Thus we shall formulate and analyze models involving multi-server “pools,” with the understanding that servers belonging to the same pool are identical.

Unfortunately, there seems to be no single term that is appropriate in all applications for what were called “materials” in the model description above. For example, one might want to model and analyze activities in which the “materials” processed are telephone calls from customers seeking different kinds of airline reservations, or to model and analyze activities in which the “materials” produced as outputs are electronic components of various types.

Because the word “material” is generally interpreted to have a meaning more narrow than required in these examples, we shall more often use the abstract term “units of flow,” or else use concrete terms like “customer,” “job,” or “packet” that relate to particular application domains. The important point here is that *units of flow will be called by different names at different points in this book.* We are confident that readers, having been alerted to this phenomenon, will find that it causes no confusion.

In his discussion of model formulation in linear programming, Dantzig (1998) described an “activity” as a black box, into which flow inputs and out of which flow outputs, defining “black box” as “any system whose detailed internal structure one willfully ignores” (page 32). In the introductory chapter of that same text, it is said that “the first step [in model formulation] consists in regarding a system under design as composed of a number of elementary functions that are called ‘activities’ ... The different activities in which a system can engage constitute its technology” (page 6).

Obviously, there is modeling discretion involved in the definition of resources, activities, and units of flow. Different levels of aggregation are possible, and the right choice depends on what decisions are to be supported or informed by the analysis.

1.3 Two Examples of Queueing Networks

Most past research on stochastic processing networks has focused on a narrow class of models called queueing networks, the general definition of which is postponed to Section 2.6. Here we consider two relatively simple examples, using the language of activity analysis to establish parallels with the classical input–output model of linear programming. This discussion provides a stepping stone to the general notion of a stochastic processing network, and allows us to introduce in a concrete setting some concepts and terminology that will be used in more general contexts later.

We begin with the same example used in the preface of this book, namely, the arrangement of two servers in tandem that is pictured in Figure 1.1. There are two servers, labeled S1 and S2 in the figure, that play the role of processing resources. Following standard usage in queueing theory, we call the units of flow *customers*, and assume that new customers arrive from outside the system according to a Poisson process with arrival rate $\lambda_1 > 0$. (The reason for the subscript 1 will become apparent shortly.) There are two storage *buffers* (or waiting rooms) in Figure 1.1, labeled B1 and B2, where customers wait for

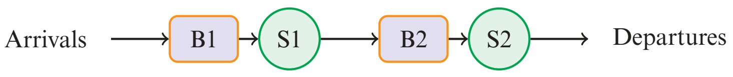

Figure 1.1 Tandem queueing network.

service if they cannot be processed immediately, and where they continue to reside as they are being processed. (The verbs "to serve" and "to process" will be used interchangeably throughout this book, as will the nouns "service time" and "processing time.")

New arrivals enter buffer 1, are processed by server 1 on a first-in-first-out (FIFO) basis, then proceed to buffer 2, are processed by server 2 on a FIFO basis, and then depart from the system. Customers who have not yet completed their first service are designated as *class 1* customers, and those who have completed their first service but not the second one are designated as *class 2*. Thus it is class 1 customers who occupy buffer 1, and class 2 who occupy buffer 2. We assume that the service times or processing times for class i customers are independent and identically distributed (i.i.d.) random variables with mean $m_i > 0$ $(i = 1, 2)$, and that the arrival process and the two service time sequences are mutually independent.

There are only two processing activities in this system, namely, the processing of class 1 by server 1 and the processing of class 2 by server 2. A crucial notion for stochastic processing networks generally is that of a *control policy*, by which we mean a rule or set of rules that determine which activities will be undertaken when, based on dynamic observations of system status. (Hereafter the term "control policy" will be routinely shortened to just "policy." It is common in the literature of applied probability to distinguish among various categories of dynamic control, such as routing, sequencing, and input control. Here the term "control," occasionally expanded to "dynamic control" for emphasis, is understood to include all such categories.) For the tandem queueing example pictured in Figure 1.1, there are no meaningful control decisions to be made, given the restriction to FIFO processing stated above: under most commonly used performance criteria, one wants each server to devote its full capacity to processing the first-arriving customer in its associated buffer, as opposed to idling the server or having it work at less than full capacity, which would simply delay completion of the customer's service.

Can we expect the system to achieve long-run stability (see Section 1.5 for an explanation of that term) with FIFO processing? The first thing to consider in that regard is the adequacy of server capacity. Because arrivals occur at an average rate of λ_1 jobs per time unit, the expected time required from server i to complete the processing of jobs that arrive within one time unit is $\lambda_1 m_i$ $(i = 1, 2)$. That product, hereafter called the *load factor* for server i, expresses the load imposed on the

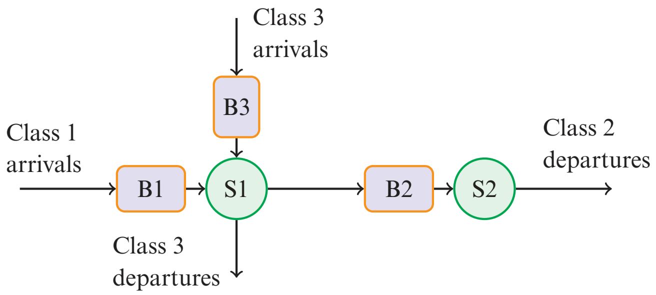

Figure 1.2 The criss-cross network.

server as a fraction of capacity, and it will later be shown that stability is achieved if and only if

(1.1) $$\lambda_1 m_1 < 1 \quad \text{and} \quad \lambda_1 m_2 < 1.$$

Figure 1.2 pictures a more complicated model, commonly called *the criss-cross network*, where there *are* meaningful control decisions to be made. As in the tandem queueing example, we have two servers and a stream of customers who require one processing operation from each server, but there is also a second stream of customers, and they require just a single processing operation from server 1 before they depart. Customers in the latter stream, whether waiting or being processed, are referred to as *class 3*, and we imagine them as being stored in their own dedicated buffer, as pictured in Figure 1.2. In addition to the assumptions stated earlier for class 1 and class 2 customers, we assume that class 3 customers arrive according to a Poisson process at rate $\lambda_3 > 0$, that their service times are i.i.d. with mean $m_3 > 0$, and that the class 3 arrival process and service time sequence are independent both of one another and of the class 1 input process and the service time sequences for classes 1 and 2.

The criss-cross network is an example of a *multiclass* queueing network, which means that there is at least one server (in this case, server 1) that has responsibility for processing two or more distinct customer classes. If we require that customers within each class be processed on a FIFO basis, which is a reasonable restriction in many settings, then the choice of a control policy amounts to specifying whether class 1 or class 3 customers are to be processed first by server 1, based on the numbers of customers currently occupying the three buffers, and possibly also on other aspects of the system's history or current status. There is no uniquely "best" control policy, because generally speaking, choices that reduce the waiting times of class 3 customers will increase waiting times for customers who require processing from both servers, and vice versa.

With regard to the potential for long-run stability, we first observe that on average, server 1 must work for $\lambda_3 m_3$ time units to process the class 3 jobs that arrive in one time unit, so the total load factor for server 1 is $\lambda_1 m_1 + \lambda_3 m_3$, while the load factor for server 2 is the same as in our tandem queueing example. Thus we have the following analog of the stability condition (1.1):

$$\lambda_1 m_1 + \lambda_3 m_3 < 1 \quad \text{and} \quad \lambda_1 m_2 < 1. \tag{1.2}$$

The criss-cross network is a *feedforward* queueing network, which means that servers can be numbered in such a way that customers never move from higher numbered servers to lower numbered ones. As a consequence, any non-idling dynamic control policy (that is, any policy that requires each server to work at full capacity except when all its associated buffers are empty) will achieve long-run stability if both inequalities in (1.2) hold. This will be proved in Chapter 8 under mild additional assumptions. In contrast, Section 1.6 provides examples of *non*-feedforward networks where long-run stability is *not* achieved by a particular nonidling policy, even though stability *is* achievable using other, more intelligent policies. Examples of this kind, which first came to light in the early 1990s, provided the motivation for much of the research recounted in this book.

1.4 SPN Examples with Additional Features

As we shall see in future chapters, multiclass queueing networks can be much more complicated than the examples discussed above, but our conception of a stochastic processing network (SPN) further allows various interesting and realistic structural features that are *not* allowed in the queueing network framework. Some of those features are illustrated by the four examples discussed in this section, each of which falls within some model family treated later in the book.

Alternate routing example. The system pictured in Figure 1.3 differs from the criss-cross network (Figure 1.2) in that either server 1 or

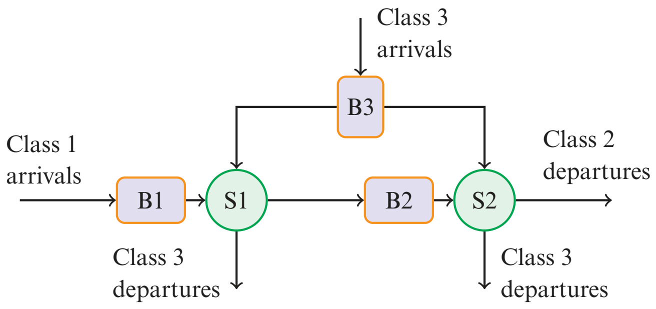

Figure 1.3 A system with alternate routing.

server 2 can provide the single processing operation needed by class 3 customers; the processing time distribution may or may not be different depending on which server is chosen. (We are assuming here that the decision about which server will process a class 3 arrival need not be made until the service is about to begin, so class 3 arrivals are held in a single buffer while awaiting service. Alternatively, one might assume that a commitment must be made at the moment of arrival; see Section 4.2 for further discussion.) In that case, there are four distinct activities: the processing of class 1 by server 1, of class 2 by server 2, of class 3 by server 1, and of class 3 by server 2. Thus a control policy must allocate the capacity of each server to two potential activities over time, based on observed system status.

In the simple example of alternate routing just described, class 3 customers depart from the system after being processed, regardless of which server does the processing. But more generally, when a customer class can be processed by any of several different servers, one may allow the future routing of such customers to depend (in the probabilistic sense) on which server is chosen. Obviously, alternate routing capabilities create a rich and complex environment for dynamic system control.

Bandwidth sharing example. Figure 1.4 pictures a network with two servers, labeled S1 and S2, that process *jobs* of three classes. Servers 1 and 2 have *capacities* b_1 and b_2, respectively, the significance of which will be explained shortly. The long-run average arrival rate for class i is λ_i, and each class i arrival has a *size* that is drawn from a class-specific distribution with mean $m_i > 0$ $(i = 1, 2, 3)$. Job sizes for the three classes

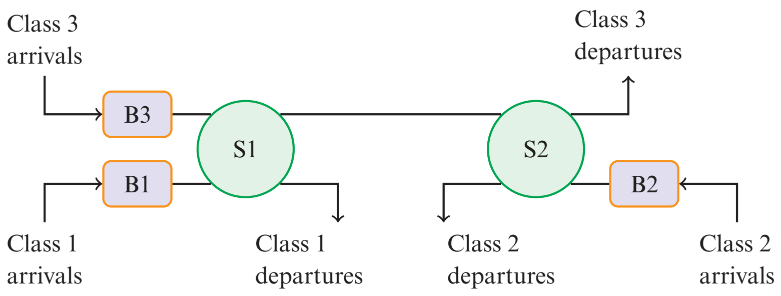

Figure 1.4 A two-link bandwidth sharing network.

form mutually independent sequences of i.i.d. random variables that are also independent of the arrival processes. For concreteness, one may assume that arrivals of the three classes occur according to independent Poisson processes.

The processing of a job is accomplished by allocating a *flow rate* to it over time: newly arriving jobs are stored in a class-specific buffer, and a job departs from the system when the integral of its allocated flow rate equals its size. Processing a class 1 job consumes the capacity of only server 1, and processing a class 2 job consumes the capacity of only server 2, but processing a class 3 job consumes the capacities of servers 1 and 2 simultaneously and at equal rates. (Expressing this last feature in generic language, it may be said that there are three processing activities in the model under discussion, one of which involves *simultaneous resource possession*.) To be more precise, if we denote by x_i the flow rate allocated to class i at a given point in time, those allocations must satisfy the following capacity constraints:

$$x_1 + x_3 \leq b_1 \quad \text{and} \quad x_2 + x_3 \leq b_2. \tag{1.3}$$

Finally, a crucial model assumption is that the flow rate x_i at any given time is *divided equally among all the class i jobs then present in the system*. The flow rate received by any one job may vary over time, depending on the contents of the three buffers and on the resource allocation policy followed, and jobs of a given class will not necessarily finish processing in the order of their arrival.

Interpreting this example as a data communication model, one may equate servers to links of a communication network, and jobs to files that require transmission over different routes: jobs of class 1 traverse only link 1, those of class 2 traverse only link 2, and those of class 3 traverse both links without intermediate storage. Job sizes might then be expressed in bits, with flow rates x_i and link capacities (or *bandwidths*) b_i expressed in bits per second. Our "equal sharing" assumption, whereby all jobs seeking transmission over a given route at a given time share equally in the flow rate allocated to that route, is in fact a common feature of communication networks, motivated by considerations of fairness. Section 4.5 describes and analyzes a class of bandwidth sharing models that generalize the example portrayed in Figure 1.4.

Data switch example. The 2×2 data switch pictured in Figure 1.5 consists of two input ports (also called *ingresses*) labeled a and b, two output ports (*egresses*) labeled c and d, and a switching fabric that connects them. (More generally, an $m \times n$ switch has m input ports and n output ports.) This system operates in discrete time, the units of time being called *timeslots*, and its units of flow are data *packets* of uniform size.

During each timeslot, a random number of packets arrive at each ingress (that random number may be zero), and packets arriving at each ingress are logically separated into two virtual buffers according to the egress through which they must exit. Thus there are four virtual buffers on the input side of the switch, one for each combination of ingress and egress. For reasons we need not go into, this system may

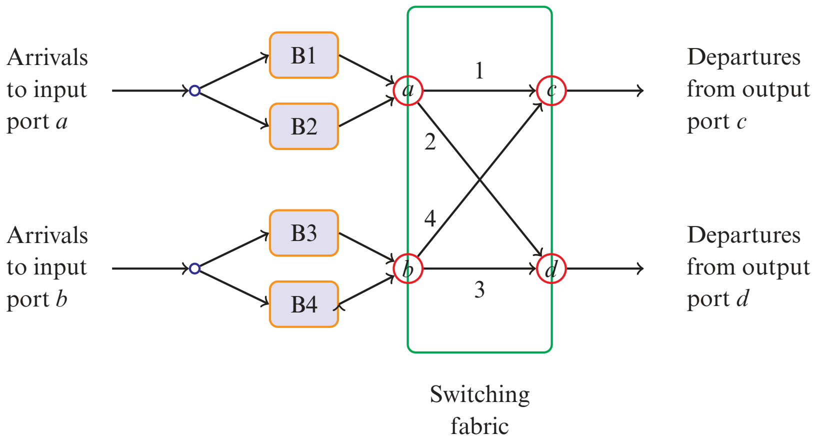
Arrivals to input port a
Arrivals to input port b
B1
B2
B3
B4
a
b
c
d
1
2
3
4
Departures from output port c
Departures from output port d
Switching fabric

Figure 1.5 A 2×2 data switch.