Contents

7

Stability of general processing networks

Jim Dai and Otis B. Jennings

Multiclass queueing networks are effective tools for capturing the dynamics of complex manufacturing systems. For instance, one can model multiple product lines as well as processes with highly reentrant flows. In addition to their industrial relevance, multiclass queueing networks present theoretical challenges absent from their single class precursors. Not surprisingly, the research community has devoted considerable effort recently to the study of such network models, both as an academic exercise as well as for practical purposes.

Still, there remain some aspects of manufacturing that escape the modelling scope of multiclass queueing networks, which we henceforth refer to as *standard networks*. Consider the following three examples. An explicit assumption of the standard network is that each work area is populated by a single server, whereas manufacturing environments often have parallel machines with similar processing capabilities. Furthermore, in many systems, servers are able to process multiple jobs simultaneously, i.e., in batches, without a negative effect on processing speed. In the standard network, servers process *one* job at a time. Lastly, consider a machine that has a job class-specific configuration, such as a baking temperature, die, mask, toolset, or physical location of personnel. Before the server can switch its processing efforts from one class to another, the associated configuration must first be set. The resulting loss in potential processing effort is referred to as a setup (delay). As far as the standard network is concerned, all setup times are assumed to be zero. As such, there is no disincentive in the standard network for excessive switching.

In response to such application shortcomings, some of the recent research efforts in queueing theory focus on extending the standard network model. In this chapter, we continue this trend by inserting all three features into a single model. That is, in our family of stochastic processing networks, we include batch processing operations, setup times and multi-server worksta-

tions. Each workstation in the model may contain any combination of these three elements. The formal model is presented in Section 7.2.

Beyond the presentation of our generalized queueing network model, the primary concern of the chapter is the stability of such networks. Our notion of stability, formalized in Section 7.2.4, is analogous to stability of standard networks. In short, a network is stable if the long-run input rate of the system is equalled by the long-run output rate. As suggested by studies of the stability of standard networks, stability for our generalized network is highly affected by the manner in which scheduling throughout the network is conducted.

Production policies govern the scheduling of servers in stochastic processing networks. Dispatch policies perform the analogous task for standard networks. The difference is that, to be effective, production policies must take the additional features of our stochastic processing network model into account. With this in mind, we define a family of "sensible" production policies that are adaptations of dispatch policies. In the following section, we justify the restriction to sensible production policies through instructive simulation examples.

In Section 7.3 we provide a framework for proving the stability of a stochastic processing network operating under a sensible production policy. Central to this framework is the artificial fluid model of a stochastic processing network. The artificial fluid model presented here is an extension of the identically termed model presented in Dai and Jennings [15], which, in turn, is a generalization of fluid models of standard networks. Having achieved widespread acceptance in the literature, fluid models of standard networks will be referred to as standard fluid models. Unlike their standard fluid model counterparts, artificial fluid models do not arise directly from a limiting procedure of some discrete network process; hence the "artificial" qualifier. Nevertheless, stability of the artificial fluid model implies stability of the stochastic processing network, a connection paralleling the relation between standard fluid models and standard networks.

Drawing on the similarities between artificial fluid models and standard fluid models, one can often exploit methods used to prove the stability of the latter to perform the same task for the former. In particular, Lyapunov functions that help demonstrate the stability of standard fluid models may also work for artificial fluid models. Unfortunately, for our most general model, this is as strong a statement as we are prepared to make. A more precise statement can be made for fluid model analogs of networks with no setup times. Such networks, referred to as batch processing networks, and their corresponding scheduling rules, referred to as full batch policies, were studied by Dai and Li [17]. Lyapunov functions and some nuances of employing them in demonstrating the stability of artificial fluid models is the focus of the discussion in Section 7.4.

As an exercise in using Lyapunov functions and the artificial fluid model framework for proving stability of stochastic processing networks, we inves-

tigate two production policies in Section 7.5. The policies, adapted from the early-steps-first and generalized round robin dispatch policies, stabilize a stochastic processing network as long as the traffic intensity at each station is less than one.

7.1 Motivating Simulations

Our performance measure of concern in this chapter is the throughput rate, or the long run rate at which jobs leave the system. As demonstrated through the well-known standard network examples of Lu and Kumar [31] and Rybko and Stolyar [33], the issue of throughput rates is not a simple matter. In this section we provide additional examples to argue that matters are further complicated in the presence of batch operations and setup times.

But first, we revisit the major takeaways from the Lu-Kumar and Rybko-Stolyar examples. In the networks under consideration therein, the nominal server utilizations are less than 100%. Yet, because of ill-conceived dispatch policies, the number of jobs in the systems grows linearly with time. Such unbounded growth in jobs hints at a discrepancy between the throughput rate and the arrival rate. Dispatch policies that produce such pathological behavior, when it is possible for the throughput rate to equal the arrival rate, are considered *inefficient*. As the examples imply, inefficient policies are not necessarily the product of needless idling of servers when jobs are present and work can be done. Notice we have yet to discuss the effect of setups and batch operations!

Setups and batch processing operations have the obvious efficiency ramifications. Nevertheless, we suggest an extra dose of caution when addressing the effects of such features. Given a server that is subject to setups, excessive switching of the processing effort between different job classes results in a significant loss in potential server effort, measured in units of time. Successful scheduling of such servers will result in sufficiently long stretches of time during which the server dedicates its full effort to a single job class. Exhaustive service is a rather extreme example of such scheduling rules-of-thumb. Under exhaustive service, the server processes jobs of the same class until there are no longer jobs of that class present.

Although this method of exhaustive service before switching is highly effective in single station systems, blind adherence to the policy can be detrimental in the network setting. Consider the following simulation study, devised by Jennings [26]. As illustrated in Figure 7.1, there are three single-server processing stations, labelled 1, 2 and 3. Each job must go through six steps, labelled 1 through 6, as they pass through the network. Jobs being processed or waiting to be processed in step k are called class k jobs. Class k jobs that are awaiting processing are said to reside in buffer k. Steps 1 and 4 are conducted at station 1, steps 2 and 5 at station 2, and steps 3

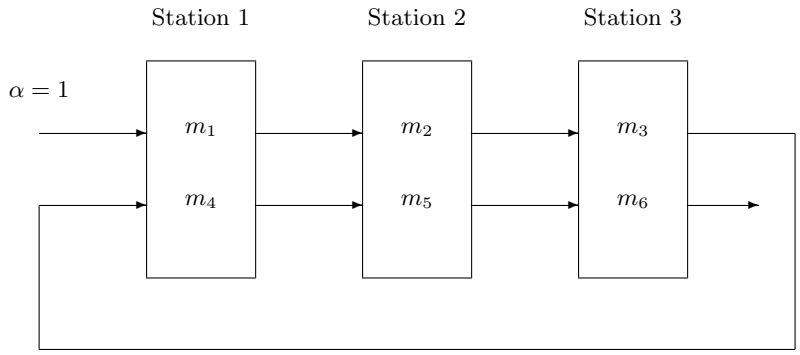

FIGURE 7.1. A three-station, six-class standard network with setups

and 6 at station 3. In the simulation, jobs arrive according to a Poisson process with rate $\alpha = 1$ job per hour. Processing times and setup times are all exponentially distributed. Processing steps 2, 4 and 6 each require 45 minutes on average. Processing steps 1, 3 and 5 require six minutes of effort on average. Setup times for a server switching between any pair of classes are 30 minutes on average. Nominally speaking, each server should be busy processing jobs a fraction

$$\frac{1 \text{ job}}{60 \text{ minutes}} \times \left(\frac{45 + 6 \text{ minutes}}{\text{job}} \right) = .85$$

of the total simulation time. Notice, this quantity is independent of setup times.

The servers employ an exhaustive service policy. Suppose a server has just processed all of the jobs of one buffer. If there are no jobs in the other buffer at the station, the server sits idle and waits for an arrival. Otherwise, the server performs a setup and then exhausts the other buffer at the station. The scheduling policy just described exhibits the so-called *non-idling* property; that is, no server idles when there are jobs at its corresponding station. The graph in Figure 7.2 depicts the total number of jobs in the system, or work-in-progress (WIP) as a function of time. One can see that the WIP exhibits a roughly linear growth trend, again a strong sign of instability.

To better understand what is going wrong in this system, we track how potential server effort is being used over the course of the simulation. The results, summarized in Table 7.1, reveal that no server is busy 85% of the time, as expected from the pre-simulation calculations. The implication is that the servers are not working enough to process the arriving jobs. What is perhaps more disturbing is the reason, or lack thereof, for why the servers are not working. It is *not* the case that servers are spending too much time performing setups. To the contrary, setup time is insignificant in terms of distribution of server effort. Neither are servers idling when jobs

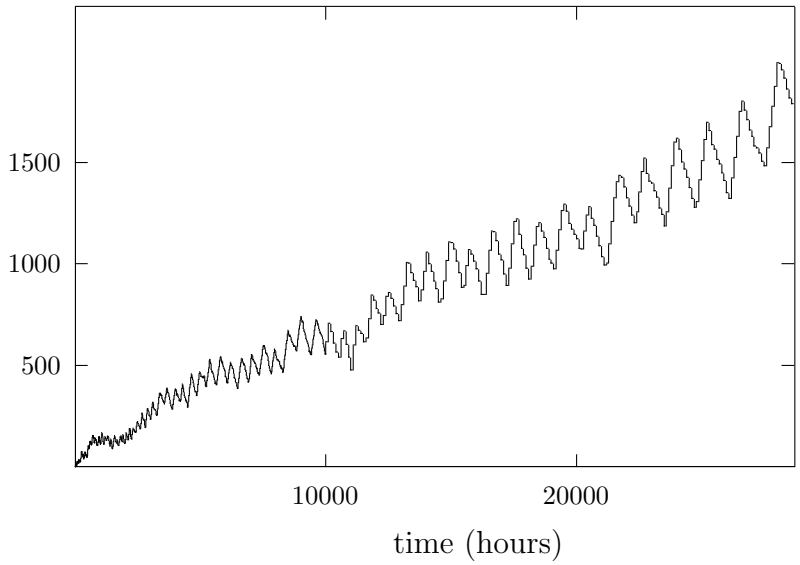

FIGURE 7.2. WIP as a function of time.

are present, as dictated by the non-idling condition.

For more evidence of what is wrong with the system, see Figure 7.3, where we plot the job counts for steps 2, 4 and 6 as functions of time. It is rarely the case that buffers 2, 4 and 6 are all simultaneously strictly positive. The implication is that, throughout the majority of the simulation, at most two of these buffers can be processed simultaneously. This fact would not be alarming if the steps were processed at two servers. But the steps reside at three distinct stations.

The behavior just described is an example of a so-called *pseudo-station*, also referred to by Bertsimas [2] as a *K-virtual station*. For more on pseudo-stations, see Hasenbein [25]. The behavior is related to an example in Dai, Hasenbein and Vande Vate [14], the origin of the network in Figure 7.1. The fundamental differences in [14] are that jobs are replaced by units of fluid and that there are no setups. Not surprisingly, further simulation studies reveal that the same pathological behavior in our discrete network

Utilization Profile			
	Server 1	Server 2	Server 3
Idle	.181	.189	.202
in-Setup	.003	.002	.002
in-Service	.816	.809	.796

TABLE 7.1. Summary of server usage.

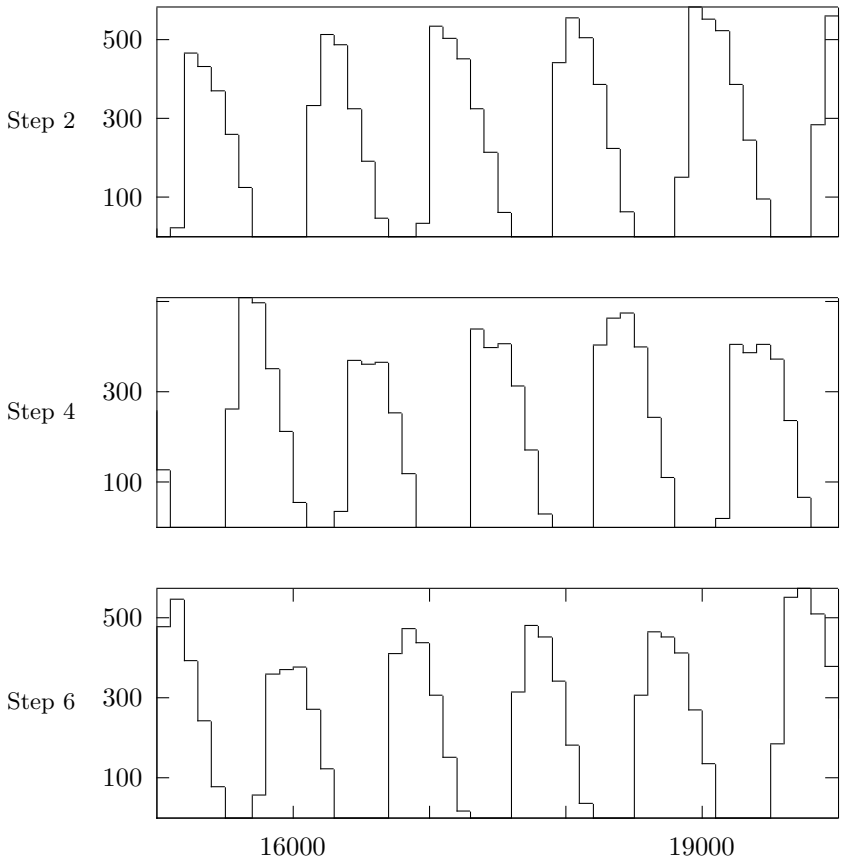

FIGURE 7.3. Job count at steps 2, 4 and 6.

can occur when the setup times are equal to zero.

In this first simulation example, exhaustive service led to instability. In many other cases, namely those in Lu and Kumar [31], Rybko and Stolyar [33], and Dai, Hasenbein and Vande Vate [14], instability may be caused by so-called static buffer priority (SBP) policies. Under an SBP policy, jobs of some buffers have strict processing priority over jobs in other buffers. Byproducts of SBP policies are long production runs during which a single class of jobs is processed. These long production runs may starve downstream buffers which, once they finally receive a very large number of jobs during a relatively short interval, ultimately starve the buffers responsible for starving them. This cycle of "starve and be starved" continues as oscillations of WIP grow larger and larger. In terms of the dynamics leading to instability, exhaustive service effectively mimics SBP policies.

One lesson to be gleaned from this example is that, in the presence of

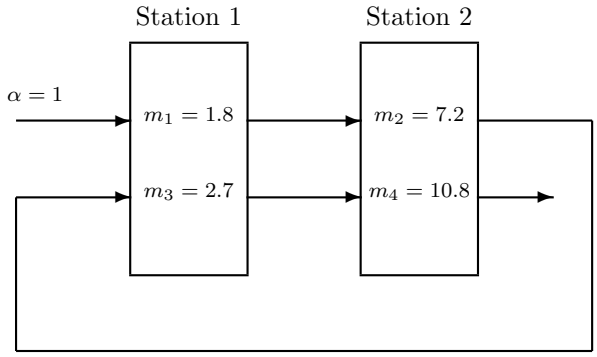

FIGURE 7.4. A two-station, four-class batch processing network

setups, the duration of dedicated server effort should be *long, but not too long*. For single-station systems with setups, or polling stations, this notion is embodied in the ℓ-limited service scheduling policy; see, for example, Takagi [34]. For such policies, the number of consecutively processed class k jobs is forced to equal ℓ_k, unless the corresponding buffer is exhausted first. Policies of this limited variety will resurface when we define our family of "sensible" policies.

A second lesson is that dispatch policies that prove successful in scheduling standard networks may inform our decision-making process when scheduling the more general processing networks. This is true whether stations are populated by setup servers, batch servers or combinations thereof. Indeed, this is the approach employed by both Dai and Jennings [15] for queueing networks with setups and Dai and Li [17] for batch processing networks.

With servers that process multiple jobs simultaneously, forming large batches is desirable. Not doing so can also be considered an inefficient use of potential server effort. Empowered by the second lesson above and equipped with a dispatch policy proven to work well for a standard network, one might reach erroneous conclusions. Specifically, one might declare that as long as a smart dispatch policy is used to select the class to be processed next and the subsequently constructed batch is as large as possible, throughput will be maximized. However, as demonstrated by Dai and Li [17], this reasoning is flawed. Consider their simulation.

The network has two single-server stations serving four job classes, as illustrated in Figure 7.4. As with the previous simulation, this network is an example of a reentrant line, where all jobs follow the same deterministic route through the network. Each job follows four processing steps, corresponding with the four classes and alternating between stations 1 and 2. The batches for steps at station 1 contain at most five jobs and the maximum batch size at station 2 is 20. Jobs arrive from the outside according to a Poisson process with rate $\alpha = 1$ job per minute. The processing times

for class k batches are independent, exponentially distributed with mean m_k, $k = 1, 2, 3, 4$. The mean service times are set to be $m_1 = 1.8$, $m_2 = 7.2$, $m_3 = 2.7$, and $m_4 = 10.8$ minutes, as shown in the figure.

As in the previous simulation, we compute the traffic intensity as the product of the arrival rate and the mean service time. In the presence of batches, the mean processing time can be amortized over all of the jobs in a maximum sized batch. Hence, the traffic intensities are given by

$$\rho_1 = \alpha(m_1 + m_3)/5 = 0.9 \quad \text{and} \quad \rho_2 = \alpha(m_2 + m_4)/20 = 0.9.$$

A more formal definition of traffic intensities will be given in (7.5) in Section 2. Clearly, the usual traffic condition (7.6) is satisfied for the parameter set. Intuitively, the batch processing network should have enough capacity to handle all incoming jobs, achieving a throughput of 1 job per minute.

The last-buffer-first-serve (LBFS) dispatch policy maximizes throughput in a standard reentrant line (i.e., without batch operations); see, for example Dai and Weiss [19] or Kumar and Kumar [29]. Under LBFS, steps 3 and 4 have priority over steps 1 and 2, respectively. It is reasonable to believe that, with a few modifications, LBFS can perform the dispatching duties for reentrant lines with batch operations. The simulation carries out this idea. Namely, once the LBFS policy selects a buffer, we process the largest batch possible. Under this modified LBFS policy, referred to as the LBFS batch policy, each server always selects jobs from the highest priority nonempty buffer to form a batch, even though the selected buffer may have only one job in it. The following table shows the average times in system.

Number of jobs leaving the system	50	500	5000	50000
Average time in system	54.2	208.4	1057.3	6831.6

Figure 7.5 plots the total number of jobs in the system as time increases. Clearly, the system is unstable, thus it cannot handle the traffic loads in long run. On the other hand, the same simulation shows that, after completing 50000 jobs, server 1 is busy 96% of the time with average batch size 4.19 jobs and server 2 is busy 99.97% of the time with average batch size 16.41 jobs. In contrast with our first simulation, the servers this time are apparently heavily utilized, yet the system is unstable. Under the LBFS batch policy, server 2 keeps serving class 4 batches that may consist of only five jobs, sent recently from class 3 by server 1. Moreover, this takes place even when class 2 has a large number of jobs waiting. This example shows that, in the presence of batch operations, a naive adaptation of a dispatch policy may prove inefficient, although the policy performs well in a standard network.

This brings us to our third and final lesson. For a batch server, it is not enough to form the largest batch *after* the class has already been chosen. Similarly, for a server subject to setups and operating under an ℓ-limited policy, it is not enough to attempt to process ℓ_k jobs once class k is already

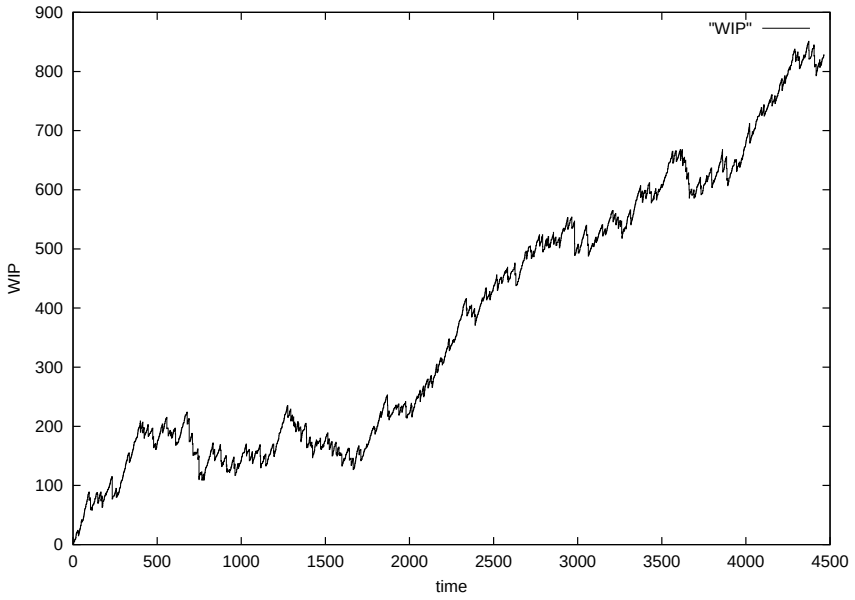

FIGURE 7.5. The total number of jobs in system

chosen. Instead, when the class is being selected, we should restrict attention to those buffers from which a sufficiently large batch can be formed, or a sufficiently long production run can be performed, if such a buffer exists.

7.2 Open processing networks

In this section we present the family of networks under study throughout the chapter. The model is an extension of open multiclass queueing networks, as presented by Harrison [23]. For the purpose of this chapter, we will refer to that model as the standard network. Our model, which adds setup times, batching operations, and multiple-server workstations to the standard network, will be referred to as the *open stochastic processing network*, or processing network for short. Readers should be warned that Harrison [24] uses that term with a somewhat different meaning. We adopt the name nonetheless.

In the remainder of the section, we derive the corresponding standard network of a given processing network. After drawing connections between scheduling the resources of both types of networks, we conclude with a formal definition of rate stability.

7.2.1 Network description

Consider a network of stations, labelled $j = 1, \ldots, J$. The stations are populated by classes, labelled $k, k' = 1, \ldots, K$, where class k is associated with a unique station $\sigma(k)$. Whenever k and j appear together, it is implied that $j = \sigma(k)$. The collection $\mathcal{C}(j)$ of all classes associated with station j is referred to as the station's constituents.

The basic processing unit is a *batch*, which consists of one or more *jobs*, the basic unit of flow. Jobs enter the network exogenously, and change classes as they move through the network. While awaiting processing, class k jobs are said to reside in buffer k. We use the terms "buffer" and "class" interchangeably. For the most part, buffer is used to connote physical location, as in a storage place for jobs awaiting processing. The class label has a more metaphysical interpretation. For instance, one removes a job from a buffer for processing, but the job retains its class designation until processing is complete.

Just prior to being processed, jobs are removed from the buffer and grouped to form a batch. Batches consist exclusively of jobs from a single class. (In some industrial settings, jobs from multiple classes can be included in the same batch, provided they share the same "recipe.") The number of jobs in a class k batch is at most B_k; the minimum is one. When a class k batch is formed, it consists of the oldest jobs in the buffer, measured in terms of their most recent arrival time to the buffer. In this sense, forming batches from jobs is first-come-first-batched. Immediately after the formation of a batch, processing commences and no additional jobs may be added to the batch. The batch grouping remains intact until processing is complete. Post-processing, the batch grouping is dissolved and the component jobs are individually routed through the network, perhaps to different buffer locations. (When a batch is dissolved, we assume an order is assigned to the departing jobs.) All jobs eventually leave the network.

Each station has a pool of servers that share the processing duties at the station. We denote the number of servers residing at station j by p_j, a quantity which generally varies from station to station. The servers of a given station are identical in the sense that each has the capability of processing any batch formed at the station. Moreover, the length of the processing time is independent of which server performs the work. A server may process at most one batch at a time and batches may be processed by at most one server. That is, servers cannot combine efforts in the processing of any batch. Once a server starts processing a batch, it cannot be interrupted. In other words, there is no preemption of service. (There may be occasions where it makes sense to allow preemption of service. We exclude it for modelling ease.) We will occasionally say that a server is processing a class or buffer, meaning, processing the batches from that class.

Suppose a server last processed class k and is about to process a batch from a different class $k' \neq k$. Before the actual batch formation can begin,

the server must perform a *setup*. That is, a delay is incurred whenever a server switches its processing efforts between classes. Setups are denoted by the pair (k, k'), where k signifies the class just processed and k' is the subsequent class. Generally, the duration of a type (k, k') setup, or *setup time*, depends on both k and k', as well as their order. In this sense, setups are *sequence-dependent*. With the inclusion of setups in our model, servers are always in one of three states: *idle*, *in-service* or *in-setup*. In addition to not being interrupted while processing batches, servers are never preempted while performing a setup.

For each class k, we have the cumulative processes $E_k = \{E_k(t), t \geq 0\}$, $V_k = \{V_k(n) : n = 1, 2, ...\}$, and $\Phi^k = \{\Phi^k(n) : n = 1, 2, ...\}$. For each time $t \geq 0$, $E_k(t)$ counts the number of external arrivals to class k in $[0, t]$. For each positive integer n, $V_k(n)$ records the total service time requirement for the first n class k batches (regardless of batch size or processing server). For each positive integer n, $\Phi^k(n)$ is a K-dimensional vector with each component being a nonnegative integer. For each class k', $\Phi^k_{k'}(n)$ records the number of the first n *processed* class k jobs that are routed to buffer k' upon completion of service. When $\Phi^k(n-1) = \Phi^k(n)$, the nth processed class k job immediately leaves the system. By convention, we assume

$$E_k(0) = 0, \quad V_k(0) = 0, \quad \text{and} \quad \Phi^k(0) = 0.$$

For each time $t \geq 0$, we extend the definitions of $V_k(t)$ and $\Phi^k(t)$ as follows:

$$V_k(t) = V_k(\lfloor t \rfloor) \quad \text{and} \quad \Phi^k(t) = \Phi^k(\lfloor t \rfloor),$$

where $\lfloor t \rfloor$ denotes the largest integer less than or equal to t.

In addition, we define the cumulative process $F_{kk'} = \{F_{kk'}(n) : n = 1, 2, \ldots\}$, associated with type (k, k') setups. When k and k' appear together in the same subscript of a setup quantity, it is implied that the classes are distinct, $k \neq k'$, and that they reside at the same station, $\sigma(k) = \sigma(k')$. For each positive integer n, $F_{kk'}(n)$ records the total time required for the first n setups from class k to k'. Again we assume $F_{kk'}(0) = 0$. Furthermore, for each $t \geq 0$, we extend the definition so that $F_{kk'}(t) = F_{kk'}(\lfloor t \rfloor)$.

We call (E, V, Φ, F) the set of primitive processes, where $E = \{E(t), t \geq 0\}$, $V = \{V(t), t \geq 0\}$, $\Phi = \{\Phi(t), t \geq 0\}$, and $F = \{F(t), t \geq 0\}$, with $E(t) = (E_1(t), E_2(t), ..., E_K(t))'$, $V(t) = (V_1(t), V_2(t), ..., V_K(t))'$, $\Phi(t) = (\Phi^1(t), \Phi^2(t), \ldots, \Phi^K(t))$, and $F(t) = \{F_{kk'}(t),\ k, k' \in \mathcal{C}(j),\ j = 1, \ldots, J\}$. When appearing with a vector quantity, a prime symbol denotes a column vector.

We assume that the strong law of large numbers holds for the primitive processes; namely, with probability one,

$$\lim_{t\to\infty} \frac{E_k(t)}{t} = \alpha_k, \quad \lim_{t\to\infty} \frac{V_k(t)}{t} = m_k,$$

$$\lim_{t\to\infty} \frac{\Phi^k_{k'}(t)}{t} = P_{kk'}, \quad \text{and} \quad \lim_{t\to\infty} \frac{F_{kk'}(t)}{t} = s_{kk'}. \tag{7.1}$$

The parameter set (α, m, P, s) with $\alpha = (\alpha_1, \ldots, \alpha_K)'$, $m = (m_1, \ldots, m_K)'$, $P = (P_{kk'})$, and $s = \{s_{kk'}, k, k' \in \mathcal{C}(j),\ j = 1, \ldots, J\}$ has the following interpretation: For each k, α_k is the external job arrival rate to buffer k and m_k is the mean service time for class k batches. (Recall that the processing time of a batch is independent of its batch size.) For classes k and k', $P_{kk'}$ is the long-run fraction of class k jobs that immediately become class k' jobs after being processed. It is also called the routing probability from class k to class k'. The $K \times K$ matrix P is called the routing matrix. We assume that the network is open, i.e., the matrix

$$Q = I + P' + (P')^2 + ...$$

is finite, which is equivalent to the fact that $(I - P')$ is invertible such that $Q = (I - P')^{-1}$. A reentrant line is a special type of processing network in which all jobs follow the same deterministic route of K steps, and jobs may visit some stations multiple times. The transition matrix of a reentrant line consists of $K-1$ ones and the remaining entries are all zeros. For each pair (k, k'), $s_{kk'}$ is the mean setup time when a server switches from class k to class k'. Let $s_k = \max_{k' \in \mathcal{C}(j)} s_{k'k}$ be the maximum possible mean setup time *to* class k.

For future purposes, we introduce the counting processes $\Psi = \{\Psi(t) : t \geq 0\}$ associated with the primitive service process V and $\Upsilon = \{\Upsilon(t) : t \geq 0\}$ associated with the primitive setup process F. For each time $t \geq 0$, $\Psi(t) = (\Psi_1(t), \ldots, \Psi_K(t))'$ with

$$\Psi_k(t) = \max\{n : V_k(n) \leq t\}, \quad k = 1, ..., K$$

and $\Upsilon(t) = \{\Upsilon_{kk'}(t),\ k, k' \in \mathcal{C}(j),\ j = 1, \ldots, J\}$ with

$$\Upsilon_{kk'}(t) = \max\{n : F_{kk'}(n) \leq t\}, \quad k, k' \in \mathcal{C}(j), \quad j = 1, \ldots, J.$$

The process $\Upsilon_{kk'}$ is only defined when $s_{kk'} > 0$. It follows from the strong law of large numbers (7.1) that

$$\lim_{t \to \infty} \frac{\Psi_k(t)}{t} = \mu_k, \quad k = 1, \ldots, K, \tag{7.2}$$

where $\mu_k = 1/m_k$, and that, for each setup pair (k, k') such that $s_{kk'} > 0$,

$$\lim_{t \to \infty} \frac{\Upsilon_{kk'}(t)}{t} = 1/s_{kk'}. \tag{7.3}$$

Let $\lambda = (\lambda_1, \ldots, \lambda_K)'$ be the vector of nominal total arrival rates. It is defined by the following system of equations

$$\lambda_k = \alpha_k + \sum_{k'=1}^{K} \lambda_{k'} P_{k'k}, \quad \text{for each} \quad k = 1, \ldots, K. \tag{7.4}$$

In vector form, $\lambda = \alpha + P'\lambda$. Since P is transient, the unique solution to (7.4) is given by $\lambda = Q\alpha$. We define the traffic intensity ρ_j for station j as

$$\rho_j = (1/p_j) \sum_{k \in \mathcal{C}(j)} \lambda_k (m_k/B_k), \quad j = 1, \ldots, J, \tag{7.5}$$

with $\rho = (\rho_1, \rho_2, \ldots, \rho_J)'$ being the corresponding vector. In the absence of batch operations ($B_k = 1$), this definition matches previous definitions of traffic intensity in the presence of multiple, identical servers; see, for example, Gross and Harris [22]. One can think of ρ_j as the average of the nominal utilizations of the p_j servers at station j if every batch is of the maximum size. Because class k batch sizes can be smaller than B_k, the fraction of actual effort dedicated to processing batches may very well exceed ρ_j. When, for each station $j = 1, ..., J$,

$$\rho_j < 1, \tag{7.6}$$

we say that the usual traffic condition is satisfied for the processing network.

7.2.2 The standard network and dispatch policies

We now define the corresponding *standard network* of a processing network. Our notational convention is to use a tilde when referring to a quantity associated with the standard network. The corresponding standard network is identical to the processing network except that (a) the maximum batch size is one, or equivalently, jobs are the basic processing unit; (b) there is only one server at each station; (c) the setup times are zero such that the server exists in either the idle state or the in-service state; and (d) the primitive service process is given by $\tilde{V}_k = \{\tilde{V}_k(n) : n = 1, \ldots\}$, where $\tilde{V}_k(n) = V_k(n)/(B_k p_j)$. As a result, the counting process $\tilde{\Psi}$ associated with the primitive service process $\tilde{V}$ is described by $\tilde{\Psi}_k(t) = \max\{n : \tilde{V}_k(n) \leq t\} = \Psi_k(B_k p_j t)$, for each $k = 1, \ldots, K$. Accordingly, the strong law of large numbers (7.1) yields

$$\lim_{n\to\infty} \frac{\tilde{V}_k(n)}{n} = \frac{m_k}{B_k p_j} \equiv \tilde{m}_k \quad \text{and} \quad \lim_{t\to\infty} \frac{\tilde{\Psi}_k(t)}{t} = B_k p_j \mu_k \equiv \tilde{\mu}_k, \tag{7.7}$$

for each $k = 1, \ldots, K$. Moreover, the traffic intensity for station j in the standard network is the same as in the processing network, i.e., $\tilde{\rho}_j = \rho_j$. It goes without saying that the usual traffic condition for the standard network holds if and only if the traffic condition for the processing network (7.6) holds. In short, in the standard network, stations process one job at a time, there is no loss in potential server effort from switching, and when class k jobs are in service, the station $j = \sigma(k)$ processing rate increases by a factor of $B_k p_j$ over a corresponding server's speed in the processing network.

For a stochastic processing network driven by the primitive processes (E, V, Φ, F) with maximum batch sizes $(B_1, \ldots, B_K)'$, the corresponding standard network is driven by the primitive processes $(E, \tilde{V}, \Phi)$ with maximum batch sizes that are equal to one.

Whenever multiple jobs reside at a station, there is discretion in the processing order of those jobs. For standard networks, the *dispatch policy* $\tilde{\pi}$ is the sole mechanism by which servers are assigned to classes. That is, when a server becomes available for processing, the dispatch policy selects the class from which the next job will be processed. Given the class assignment, the oldest job, based on arrival to the associated buffer, is processed. In this sense, jobs within a single buffer are processed in a first-come-first-served (FCFS) fashion. A dispatch policy is said to be *non-idling* if a server is never in the idle state when jobs are present at the station.

Not surprisingly, scheduling of servers in a standard network is less complex than in a processing network. To stress this point, an alternative term is used to distinguish between the two scheduling tasks. A *production policy* is to the processing network what a dispatch policy is to a standard network. One of the main themes of this chapter is that dispatch policies that work well for standard networks are useful in crafting effective production policies for processing networks.

7.2.3 Production policies and sensible policies

The decision process governing the formation and processing of batches in a processing network is embodied in the *production policy*. Because of the complex nature of each station, we envision most "useful" production policies having the following three-tiered approach. When a server requires an assignment to a class for processing, one first filters the set of constituent classes into a subset of *eligible* classes. Secondly, the server is *dispatched* to one of the eligible classes. The final decision involves setting the termination time of the assignment as well as determining how batches are formed in the interim.

Throughout this chapter, we assume production policies have the form $\pi = (\theta, \tilde{\pi}, \ell)$. (Exceptions are discussed in Section 7.6.) The K-dimensional vectors $\theta = (\theta_1, \theta_2, \ldots, \theta_K)$ and $\ell = (\ell_1, \ell_2, \ldots, \ell_K)$ of positive integers enforce the filtering function. When a server at station j requires an assignment, the constants determine which constituent classes are eligible. Class k is eligible if the number of class k jobs is equal to or greater than

$$\theta_k + \ell_k B_k \times \text{ the \# of servers already assigned to class } k.$$

(Exceptions that relax the eligibility requirement can be found in Section 7.6.) It is possible that the collection of eligible classes at station j is empty. In cases where no class passes the first eligibility test, the criterion is relaxed. Under the relaxed test, any class with a nonempty buffer is eligible. From the standpoint of the first eligibility test, if the number of class

k jobs is less than θ_k and no server is assigned to it, that class is effectively empty. In this sense, the components of θ can be thought of as *thresholds*.

The second component of the production policy π is the dispatch policy $\tilde{\pi}$. This terminology is borrowed from literature on standard networks. The idea here is the same. Given the current state of the system, dispatch policies perform the actual assignment of servers to classes. With the standard network, the assignment lasts at most through the processing of a single job. For processing networks, however, the assignment lasts (potentially) for the processing of several batches. We assume that the largest possible batches are formed and processed until a new assignment is sought.

The stretch of time during which the dispatched assignment holds is referred to as the *production run*. The length of the production run or, equivalently, the number of batches processed before seeking a new assignment, is the third and final decision to be made. The vector ℓ determines the length of the production run. If the assignment is for a class k that passed the first eligibility test, then we process ℓ_k batches before seeking a new assignment (if possible). If the class only passed the relaxed eligibility test, we form and process the largest possible batch and terminate the production run immediately, allowing the opportunity to make another production run decision based on updated system information.

The non-idling property for processing networks is not as straightforward as for the standard network. Part of the complication stems from having multiple servers in one of three states. As the name suggests, non-idling now refers to both the in-service and in-setup states. A server in the processing network is said to be *busy* if it is in-service or in-setup. A production policy is said to be *non-idling* if each server is allowed to idle only when there are either no jobs at the corresponding station or all of the jobs present are currently being processed in batches. (This is not to say that non-idling policies are ideal. Indeed, in some instances it may beneficial to delay processing the available jobs until a larger batch can be formed.) A consequence of the non-idling condition is that the number of busy station j servers at time t is at least

$$\min\left(p_j, \sum_{k\in\mathcal{C}(j)} \lceil Z_k(t)/B_k \rceil\right), \tag{7.8}$$

where $\lceil x \rceil$ is the smallest integer greater than or equal to x and $Z_k(t)$ denotes the number of class k jobs at time t. More concisely, a server can idle only when the constituent buffers are empty. A precise mathematical description would require knowledge of the number of jobs in each buffer k (excluding those in service), whereas we only track the aggregate class k job count process Z_k; for more, see Section 7.3.1.

The non-idling condition does not rule out small batches or short production runs, whose occurrences are limited by the eligibility requirements. A natural quantity of interest is the maximum number of servers at a given

station that can simultaneously be assigned to eligible classes, a reflection of how efficiently the resident servers must work. Consider the following index:

$$I_j(t) = \min\left(p_j, \sum_{k\in\mathcal{C}(j)} \left(1_{\{Z_k(t)\geq\theta_k\}} + \lfloor (Z_k(t)-\theta_k)^+/(\ell_k B_k)\rfloor\right)\right), \quad (7.9)$$

for each $j = 1, \ldots, J$, where $1_{\{\cdot\}}$ is the indicator function, $(x)^+ = \max(0, x)$ and $\lfloor x\rfloor$, as stated earlier, is the integer part of x. Suppose a server from station j is free at time t and requires dispatching. If $I_j(t) = p_j$ then this server will be assigned to a class that passes the more stringent eligibility test. To see this, first notice that if the class k jobcount exceeds its threshold ($Z_k(t) \geq \theta_k$) then at least one server can be assigned to this class. To each class k, for which $Z_k(t) \geq \theta_k$ holds and to which no server has been dispatched, there is the potential for assigning a server. Furthermore, for each additional $\ell_k B_k$ class k jobs, an additional server can be assigned. Therefore, $I_j(t)$ bounds from below the number of servers that *can* be assigned to eligible classes at time t. We offer a tighter bound for the number of servers processing eligible classes in Section 7.3.1.

Assuming that a server is assigned to a class k that passed the first round eligibility test, we are assured that at least $\min(\theta_k, \ell_k B_k)$ class k jobs are available at the beginning of the production run to be processed in one of at least $\min(\lceil \theta_k/B_k\rceil, \ell_k)$ batches. If we assume the thresholds are set such that they exceed the maximum number of jobs processed in a production run, i.e.,

$$\theta_k \geq \ell_k B_k, \quad (7.10)$$

then the above quantities $\min(\theta_k, \ell_k B_k)$ and $\min(\lceil \theta_k/B_k\rceil, \ell_k)$ reduce to $\ell_k B_k$ and ℓ_k, respectively. One can amortize the setup time incurred from switching to class k over all of the batches processed during the subsequent production run. The result is a *setup-adjusted mean processing time* and *rate*, which, when (7.10) holds, are respectively equal to

$$\breve{m}_k = m_k + s_k/\ell_k \quad \text{and} \quad \breve{\mu}_k = 1/\breve{m}_k, \quad k = 1, \ldots, K. \quad (7.11)$$

With the adjusted mean processing times, one can compute a *setup-adjusted traffic intensity*

$$\breve{\rho}_j = (1/p_j) \sum_{k\in\mathcal{C}_j} \lambda_k(\breve{m}_k/B_k), \quad j = 1, \ldots, J. \quad (7.12)$$

Throughout this chapter, the convention $\breve{x}$ signifies an inflation of the quantity x or simply an adjustment to the quantity x related to the influence of setups. The inflation can be due to the amortization of setup time, the presence of multiple servers, or the performance of batching operations.

We conclude with a further restriction of the type of production policies under consideration. A production policy is said to be *sensible* if it is non-idling, equation (7.10) holds, and $\breve{\rho}$ obeys the usual traffic condition; i.e.,

$$\breve{\rho}_j < 1, \quad \text{for each } j = 1, \ldots, J. \tag{7.13}$$

The reasoning behind the restriction to sensible policies is straightforward. By ensuring that the servers avoid spending an inordinate amount of time performing setups, sensible policies eliminate setups (and non-maximal batches) as sources of instability. Furthermore, implicit in the sensible policy condition is the ability to make tradeoffs of production run lengths. For instance, it is simple to compensate for setting a relatively short production run length for one class with a long production run for another class at the same station. One might use such tradeoffs as a mechanism for accounting for high priority "hot lots," for which production run length (and batch size) considerations are secondary at best. The last reason for restricting to sensible policies is more subtle. Having the non-idling condition and (7.13) ensures that no job can be ignored indefinitely. Moreover, there exists a finite constant such that the number of jobs at a station falls below the constant infinitely often. This last feature helps to demonstrate positive Harris recurrence, a stronger notion of stability investigated by Dai [12] for standard networks and Jennings [27] for queueing networks with setups. The latter also employed a similar notion of sensible policies. Lastly, we point out that, unlike the analogous condition for standard networks, the traffic condition part of the sensible policy definition (7.13) is not necessary for stability; for more details, see Section 7.6.

7.2.4 Rate stability

We now define rate stability for open stochastic processing networks. Let $D_k(t)$ denote the number of jobs in the processing network that have departed class k during the interval $[0, t]$. In the following definition, the term *state* is used. The precise definition of a state depends on the particular production policy used. The system state typically includes, but is not limited to, the number of jobs in each class, the status and assignment of each server, the remaining processing times and the sizes of the batches being processed, the remaining interarrival times for jobs arriving from outside, the lengths so far of the current production runs, and the remaining setup time for each server. We do not attempt a precise definition of state here. Roughly speaking, a state is a snapshot of the network at any given time. It should contain enough information such that once the current state of the network is given, the future evolution of the network is completely determined in distribution. Readers are referred to Dai [12] and Bramson [5] for examples and additional discussions of states in standard networks under various dispatch policies.

Definition 7.1 *A processing network is* rate stable *if, for each fixed initial state, with probability one,*

$$\lim_{t\to\infty} \frac{D_k(t)}{t} = \lambda_k, \quad \textit{for each} \;\; k = 1, ..., K. \tag{7.14}$$

The processing network is rate stable if the throughput rate or departure rate from each class is equal to the nominal total arrival rate to that class. Rate stability has been advanced by Stidham and his co-authors (see El-Taha and Stidham [21] and references therein). This notion of stability was first introduced for the standard network setting in Chen [7]. As in a standard network, the usual traffic condition is necessary for rate stability of a processing network; see Dai [13]. As stated earlier, even though we specify that the setup-adjusted traffic intensities for sensible policies must obey the usual traffic condition, this is not necessary for stability of the processing network. There are other definitions of stability, such as positive Harris recurrence; see Dai [12]. The results in this chapter can be extended to those settings as well.

As mentioned earlier, the main message of this chapter is that, in some cases, a dispatch policy of a standard network can be transformed into an efficient production policy. A precise statement to this effect cannot be articulated in general. However, Dai and Li [17] successfully provide a precise result for a subclass of dispatch policies adapted for networks with zero setup times, that is, for so-called *batch processing networks*. Their result, demonstrated for networks of single-server stations, also holds for networks of multi-server stations. We will resume this discussion on batch processing networks in Section 7.4.1.

7.3 Network and fluid model equations

In this section, we define fluid models of both processing networks and standard networks. Fluid models are continuous, deterministic analogs of discrete networks and are defined through a set of equations. To describe the fluid models, we start with equations governing the dynamics of the discrete networks. Unless explicitly stated otherwise, we assume that the processing network is operated under a sensible production policy π and the standard network is operated under a non-idling dispatch policy $\tilde{\pi}$.

7.3.1 Network dynamics

The dynamics of the processing network can be captured by the process $\mathbb{X} = (A, D, S, T, U, Y, Z)$. The components $A = \{A(t), t \geq 0\}$, $D = \{D(t), t \geq 0\}$, $T = \{T(t), t \geq 0\}$, $S = \{S(t), t \geq 0\}$, and $Z = \{Z(t), t \geq 0\}$ are K-dimensional. For each class k, $A_k(t)$ denotes the number of jobs that have

arrived to class k (from external and internal sources) in $[0,t]$, $D_k(t)$ denotes the number of jobs that have departed from class k in $[0,t]$, $S_k(t)$ denotes the amount of time that servers at station $j=\sigma(k)$ have *collectively* spent setting up for class k during the interval $[0,t]$, $T_k(t)$ denotes the amount of time that the servers at station $j=\sigma(k)$ have *collectively* spent processing class k batches during interval $[0,t]$, and $Z_k(t)$ denotes the total number of class k jobs that are buffered or being served at station j at time t. (One should keep in mind that the servers at a given station work as a team. Even though servers individually dedicate portions of their potential effort to the processing of batches and the performance of setups, the quantities S and T reflects the total effort among all of the servers.) The processes A, D, S, T, and Z are called the arrival, departure, setup allocation, service allocation, and jobcount processes, respectively. The components $U=\{U(t),t\geq 0\}$ and $Y=\{Y(t),t\geq 0\}$ are J-dimensional. For each station j, $U_j(t)$ denotes the total number of jobs at station j that are buffered or being served at time t, and $Y_j(t)$ denotes the collective total amount of time that servers at station j have been idle in the time interval $[0,t]$. (Again, idle time is summed over all of the resident servers at station j.) The process Y is called the cumulative idle time process. The process $\mathbb{X}=(A,D,S,T,U,Y,Z)$ satisfies the following set of equations:

$$A(t)=E(t)+\sum_k \Phi^k(D_k(t)),\quad t\geq 0, \tag{7.15}$$

$$Z(t)=Z(0)+A(t)-D(t),\quad t\geq 0, \tag{7.16}$$

$$Z(t)\geq 0,\quad t\geq 0, \tag{7.17}$$

$$U(t)=CZ(t),\quad t\geq 0, \tag{7.18}$$

$$C(S(t)+T(t))+Y(t)=pt,\quad t\geq 0, \tag{7.19}$$

$$Y_j(t)\text{ can increase only if }\sum_{k\in\mathcal{C}(j)}\lceil Z_k(t)/B_k\rceil<p_j, \tag{7.20}$$

$$\text{additional equations associated with the particular production policy }\pi. \tag{7.21}$$

Here C is the constituency matrix defined as

$$C_{jk}=\begin{cases}1 & \text{if } k\in\mathcal{C}(j),\\ 0 & \text{otherwise,}\end{cases}$$

and $p=(p_1,p_2,\ldots,p_J)'$ denotes the J-dimensional vector indicating the number of servers at each station.

We provide a brief interpretation of equations (7.15)–(7.21); where convenient, the interpretation is component-wise. Equation (7.15) implies the cumulative arrivals to buffer ℓ consists of those jobs arriving to ℓ from the outside ($E_\ell(t)$), and those jobs routed to class ℓ after being processed in some other class. As for (7.16), the class ℓ jobcount process at time t,

$Z_\ell(t)$, is equal to the number of class ℓ jobs present initially, plus all jobs that have arrived to buffer ℓ thus far, net those class ℓ jobs that have been processed. Expression (7.17), referred to as the nonnegativity constraint, is self-explanatory. For each j, the station-wide jobcount quantities are computed in (7.18). Equation (7.19) tracks, for each station j, how the total server time has been distributed, up until time t, i.e., between performing setups, processing batches, and idling. Note that the non-idling condition (7.20) contains the expression from (7.8). For additional interpretation, a version of (7.20) appeared in Chen and Shanthikumar [9]. Finally, the production policy π, used to govern the scheduling of servers, will have a major effect on system dynamics, hence the provision in (7.21).

Condition (7.20) gives an adequate, if not completely accurate, account of what we actually mean by non-idling. However, it may be more interesting to consider the number of servers assigned to eligible classes at time t. A first pass at this quantity was presented in (7.9), where, for each $j = 1, \ldots, J$,

$$I_j(t) = \min\left(p_j, \sum_{k \in \mathcal{C}(j)} \left(1_{\{Z_k(t) \geq \theta_k\}} + \lfloor (Z_k(t) - \theta_k)^+ / (\ell_k B_k) \rfloor\right)\right).$$

We can tighten the definition of non-idling through the introduction of the K-dimensional process Z^*, referred to as the "uncommitted" jobcount process. When a server is dispatched to buffer k it reserves $\ell_k B_k$ jobs. Thus the uncommitted number of class k jobs Z_k^* decreases by $\ell_k B_k$ upon receiving a dispatched server. The uncommitted jobcount processes increases with arrivals, just as Z_k increases. Some extra care must be taken when a server is dispatched to a class with an insufficient number of uncommitted jobs. Assuming this special case is handled properly, the new non-idling condition reads

$$Y_j(t) \text{ does not increase if } Z_k^*(t) \geq \theta_k \quad \text{for some} \quad k \in \mathcal{C}(j).$$

As mentioned earlier, the selection of jobs in the formation of batches follows a first-come-first-batched order. That is, jobs that have been in buffer k the longest have priority when batches are formed. When there are multiple servers at a station, batches from the same class may be processed in parallel. If the processing time of batches from the same class differs, it is possible for the departures of jobs from a class to violate the first-in-first-out protocol that the formation of batches obeys. To account for this feature, we have the following additional equations. For $0 \leq t_1 < t_2$ and $k = 1, \ldots, K$,

$$\Psi_k(T_k(t_2)) - \breve{\Psi}_k(T_k(t_1)) \leq \frac{1}{B_k}(D_k(t_2) - D_k(t_1)) + p_j - 1 \tag{7.22}$$

when $I_j(s) = p_j$ for $s \in [t_1, t_2]$, where $j = \sigma(k)$, and

$$\breve{\Psi}_k(T_k(t_2)) - \Psi_k(T_k(t_1)) \geq \frac{1}{B_k}(D_k(t_2) - D_k(t_1)) - p_j + 1. \tag{7.23}$$

The function $\breve{\Psi}_k(t)$, defined fully in the appendix, is an inflated version of $\Psi_k(t)$. The inflated process $\breve{\Psi}_k(t)$ reflects the fact that the completion sequence of processed class k batches may diverge from the batch-formation sequence. However, as expressed formally in Lemma 7.19 of the appendix, under fluid scaling, the difference between $\breve{\Psi}_k$ and Ψ_k is negligible. Equations (7.22) and (7.23) follow from (7.75) and (7.76), both found in the appendix. For the lower bound on departures provided by (7.22), the condition $I_j(s) = p_j$ for each $s \in [t_1, t_2]$ ensures that servers at station j are making the largest batches possible. The upper bound on departures embodied in (7.23) captures the fact that class k batch sizes are no greater than B_k.

When enough jobs are present at the station, a sensible production policy restricts the frequency of class k setups to one for every ℓ_k batches. To facilitate capturing the restriction mathematically, we expand the definition of $S_k(t)$ to $S_{k'k}(t)$, the cumulative server time spent performing setups from k' to k. Clearly $S_k(t) = \sum_{k'} S_{k'k}(t)$. Suppose (7.10) holds. Then for every $t_1 < t_2$ such that $Z_k(s) \geq \theta_k + (p_j - 1)\ell_k B_k$ for every $s \in [t_1, t_2]$ we have

$$\begin{aligned} &D_k(t_2) - D_k(t_1) \\ &\quad \geq \ell_k B_k \left(\textstyle\sum_{k' \in \mathcal{C}(j)} \left[\Upsilon_{k'k}(S_{k'k}(t_2)) - \Upsilon_{k'k}(S_{k'k}(t_1)) \right] - 2p_j \right), \end{aligned} \tag{7.24}$$

for each $k \in \mathcal{C}(j)$ and $j = 1, \ldots, J$. The left hand side of (7.24) is the number of processed jobs within the interval. Because class k is eligible throughout the interval, each occurrence where both a setup for class k and the subsequent production run lie entirely within the interval, the production run consists of exactly ℓ_k batches, each with the maximum B_k number of jobs. Hence the $\ell_k B_k$ term on the right hand side. The $2p_j$ term accounts for the fact that, for each server in the pool, a setup could be in process for class k both at time t_1 and at time t_2. The setups at the beginning of the time horizon could have been initiated when no class passed the first (stricter) eligibility test. Hence, the setup is not necessarily followed by the processing of ℓ_k batches. The setups at the end of the time interval may not have a chance to be followed by the ℓ_k batches because of running out of time. We call equations (7.15)-(7.24) the *processing network equations*. Note that S, T and Y are continuous, and that A, D, and Z are right continuous with left limits. All variables are nonnegative in each component, with A, D, S, T, and Y being non-decreasing. By assumption,

$$A(0) = D(0) = S(0) = T(0) = Y(0) = 0.$$

For each processing network driven by (E, V, Φ, F), the corresponding standard network driven by $(E, \tilde{V}, \Phi)$ has similar processes. In contrast with the processing network process $\mathbb{X} = (A, D, S, T, U, Y, Z)$, the standard network process is denoted $\tilde{\mathbb{X}} = (\tilde{A}, \tilde{D}, \tilde{T}, \tilde{U}, \tilde{Y}, \tilde{Z})$. (Again, the tilde is specific to standard networks.) Note that $\tilde{S}$ is missing from $\tilde{\mathbb{X}}$ because

there are no setups in the standard network. The equations governing the standard network process are almost the same as the ones for processing network. The exceptions are that equation (7.19) lacks the $S(t)$ term and the vector p is replaced by the J-dimensional vector $e = (1, \ldots, 1)'$; equation (7.20) is replaced by

$$Y_j(t) \text{ can only increase if } U_j(t) = 0; \tag{7.25}$$

equations (7.22) and (7.23) are reduced to

$$\tilde{\Psi}_k(\tilde{T}_k(t)) = \tilde{D}_k(t), \text{ for all } t \geq 0, \ k = 1, \ldots, K,$$

which is well-known for standard networks operating under a head-of-line dispatch policy; equation (7.24) is removed; and equation (7.21) is replaced by

$$\text{additional equations associated with the particular dispatch policy } \tilde{\pi}. \tag{7.26}$$

7.3.2 Fluid models

Let $\hat{\mathbb{X}} = (\hat{A}, \hat{D}, \hat{T}, \hat{U}, \hat{Y}, \hat{Z})$ be the formal deterministic analog of the standard network process $\tilde{\mathbb{X}} = (\tilde{A}, \tilde{D}, \tilde{T}, \tilde{U}, \tilde{Y}, \tilde{Z})$. Consider the following collection of equations:

$$\hat{A}(t) = \alpha t + P' \hat{D}(t), \quad t \geq 0, \tag{7.27}$$
$$\hat{Z}(t) = \hat{Z}(0) + \hat{A}(t) - \hat{D}(t), \quad t \geq 0, \tag{7.28}$$
$$\hat{Z}(t) \geq 0, \quad t \geq 0, \tag{7.29}$$
$$\hat{U}(t) = C\hat{Z}(t), \quad t \geq 0, \tag{7.30}$$
$$C\hat{T}(t) + \hat{Y}(t) = et, \quad t \geq 0, \tag{7.31}$$
$$\hat{Y}_j(t) \text{ can increase only if } \hat{U}_j(t) = 0, \quad j = 1, ..., J, \tag{7.32}$$
$$\hat{D}_k(t) = \tilde{\mu}_k \hat{T}_k(t), \quad k = 1, \ldots, K, \tag{7.33}$$
$$\text{additional equations associated with the particular, dispatch policy } \tilde{\pi}, \tag{7.34}$$

where $\tilde{\mu}_k$ is the class k service rate for the corresponding standard network, defined in (7.7), and, as before, e is the J-dimensional column vector of 1's. Equations (7.27)-(7.34), which define the standard fluid model, are referred to as the standard fluid model equations. As with the equations describing the processing network and the corresponding standard network, we assume that the the components of the processes $\hat{T}$ and $\hat{Y}$ are zero at time zero and are nondecreasing thereafter. Any process $\hat{\mathbb{X}} = (\hat{A}, \hat{D}, \hat{T}, \hat{U}, \hat{Y}, \hat{Z})$ satisfying (7.27)-(7.34) is called a standard fluid model solution. The component processes $\hat{A}$, $\hat{D}$, $\hat{T}$, and $\hat{Z}$ are called the fluid arrival, departure, service

allocation, and buffer level processes, respectively. The quantity $\hat{U}_j(t)$ denotes the total amount of fluid at station j at time t. The process $\hat{Y}_j$ is referred to as the server idle time for station j. Standard fluid models and their solutions are fairly well-known; see, for instance, Dai [13]. Such models arise from taking fluid limits of standard networks, a topic we explore in the following section.

One can make qualitative comparisons between the standard fluid model and the equations governing the standard network. The major difference is that jobs are (discrete) units of flow in the standard network whereas flow in the fluid model is continuous; hence, the term "fluid." Along these same lines, the non-idling condition in the fluid model states that the station j cumulative idling process $\hat{Y}_j$ cannot increase in the presence of any positive amount of fluid at the station, as opposed to any jobs in the corresponding standard network equation.

Next, we present a generalization of the standard fluid model. Consider the process $\breve{\mathbb{X}} = (\breve{A}, \breve{D}, \breve{T}, \breve{U}, \breve{Y}, \breve{Z})$ and the following set of equations:

$$\breve{A}(t) = \alpha t + P'\breve{D}(t), \quad t \geq 0, \tag{7.35}$$

$$\breve{Z}(t) = \breve{Z}(0) + \breve{A}(t) - \breve{D}(t), \quad t \geq 0, \tag{7.36}$$

$$\breve{Z}(t) \geq 0, \quad t \geq 0, \tag{7.37}$$

$$\breve{U}(t) = C\breve{Z}(t), \quad t \geq 0, \tag{7.38}$$

$$C\breve{T}(t) + \breve{Y}(t) = pt, \quad t \geq 0, \tag{7.39}$$

$$\breve{Y}_j(t) \text{ can increase only if } \breve{U}_j(t) = 0, \quad j = 1, ..., J, \tag{7.40}$$

$$\breve{D}_k(t_2) - \breve{D}_k(t_1) \leq B_k\mu_k(\breve{T}_k(t_2) - \breve{T}_k(t_1)), \quad t_1 < t_2, \forall k, \tag{7.41}$$

$$\breve{D}_k(t_2) - \breve{D}_k(t_1) \geq B_k\breve{\mu}_k(\breve{T}_k(t_2) - \breve{T}_k(t_1)) \tag{7.42}$$
$$\text{if } \breve{U}_j(s) > 0 \ \forall s \in [t_1, t_2], \quad 0 \leq t_1 < t_2,$$

$$\text{additional equations associated with the particular} \tag{7.43}$$
$$\text{production policy } \pi,$$

where $\breve{\mu}_k$ is the setup-adjusted quantity, defined in (7.11), and, as before, $p = (p_1, \ldots, p_J)'$. Equations (7.35)-(7.43) are called artificial fluid model equations, and they define the *artificial fluid model* of the stochastic processing network. Any process $\breve{\mathbb{X}} = (\breve{A}, \breve{D}, \breve{T}, \breve{U}, \breve{Y}, \breve{Z})$ satisfying (7.35)-(7.43) is called an artificial fluid model solution. As with the standard fluid model, the components of $\breve{T}$ and $\breve{Y}$ are initially zero at time zero and are nondecreasing for all $t > 0$.

Although the connection between the standard fluid model and the standard (discrete) network is straightforward, we cannot claim a direct derivation of the artificial fluid model from a limiting procedure on the processing network. The dubious origin of this particular fluid model partially explains the moniker "artificial." In fact there are additional peculiarities yet to be resolved. In particular, notice the setup allocation process S of the processing network is conspicuously missing from the artificial fluid model coun-

terpart. Yet another questionable feature of the model is the component process $\breve{T}$, which, in some way, is analogous to the service allocation process T as well as the setup allocation process S. With intentional ambiguity, we refer to $\breve{T}$ as the artificial *server* allocation process. Justification of the artificial fluid model is delayed until the following section. The remaining processes, $\breve{A}$, $\breve{D}$, $\breve{Z}$, $\breve{U}$, and $\breve{Y}$, retain their interpretations and terms from the standard fluid model.

Even without a formal justification of the model, it is nevertheless possible and, more importantly, instructive to interpret the equations of the artificial fluid model. Especially interesting are those equations in the artificial fluid model that differ from their analogous standard fluid model counterparts. For instance, compare equations (7.31) and (7.39). The difference is that by time t each station in the standard fluid network has t units of potential server effort, whereas station j in the artificial fluid network has $p_j t$ potential units of *artificial* server effort. Secondly, equation (7.33) in the standard fluid model is replaced by (7.41) and (7.42) in the artificial fluid model. In the standard fluid model, departures are directly proportional to server effort. However, in the artificial fluid model, the returns on artificial server effort are not as straightforward. Equation (7.41) gives the maximum rate at which effort is converted to departing units of fluid. There is an analogous interpretation from the (discrete) processing network. To see this, consider the average rate at which jobs depart a class when maximum-sized batches are being processed and no setup delays are incurred. Equation (7.42) provides a lower bound on the departure rate of fluid as a function of the rate of artificial server allocation when there is positive fluid at the station. This particular interpretation is related to the enforcement of the sensible policy rules. For one, maximum-sized batches are being formed. Secondly, at least ℓ_k batches are being processed per class k setup. Hence, μ_k is replaced by its setup-adjusted quantity $\breve{\mu}_k = 1/\breve{m}_k \leq \mu_k$, as defined in (7.11). Finally, additional standard fluid model equations associated with the dispatch policy $\tilde{\pi}$, (7.34), may differ from artificial fluid model equations associated with the production policy π, (7.43).

Definition 7.2 *An artificial fluid model is said to be* weakly stable *if for each artificial fluid model solution* $\breve{\mathbb{X}}$ *with* $\breve{Z}(0) = 0$, $\breve{Z}(t) = 0$ *for* $t \geq 0$.

Weak stability of a standard fluid model can be defined similarly; see, for example, Chen [7].

7.3.3 Connection between processing networks and fluid models

The criterion for including an equation in the standard fluid model is that the equation is satisfied by a *fluid limit*. As suggested in the previous sec-

tion, the inclusion of an equation in the artificial fluid model has an additional step. In this section we provide the details of both the fluid limits and the additional steps.

A fluid limit of a standard network is obtained through a law-of-large-numbers limiting procedure on the standard network process. Identically, a fluid limit of a processing network is obtained through a law-of-large-numbers limiting procedure on the processing network process. Note that the processing network process $\mathbb{X}$ (resp. standard network process $\tilde{\mathbb{X}}$) is random, depending on the sample path ω in an underlying probability space. To denote such dependence explicitly, we sometimes use $\mathbb{X}(\omega)$ to denote the network process with sample path ω. For an integer d, $\mathbb{D}^d[0,\infty)$ denotes the set of functions $x : [0,\infty) \to \mathbb{R}^d$ that are right continuous on $[0,\infty)$ and have left limits on $(0,\infty)$. An element x in $\mathbb{D}^d[0,\infty)$ is sometimes denoted by $x(\cdot)$ to emphasize that x is a function of time. For each ω, $\mathbb{X}(\omega)$ is an element in $\mathbb{D}^{5K+2J}[0,\infty)$.

For each $r > 0$, define

$$\bar{\mathbb{X}}^r(t,\omega) = r^{-1}\mathbb{X}(rt,\omega) \quad t \geq 0. \tag{7.44}$$

Again, note that for each $r > 0$, $\bar{\mathbb{X}}^r(\cdot,\omega)$ is an element in $\mathbb{D}^{5K+2J}[0,\infty)$. The scaling in (7.44) is called the fluid or law-of-large-numbers scaling.

Definition 7.3 *A function $\bar{\mathbb{X}} \in \mathbb{D}^{5K+2J}[0,\infty)$ is said to be a* fluid limit *of the processing network if there exists a sequence $r_n \to \infty$ and a sample path ω satisfying (7.1) such that*

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

where, throughout this chapter, the convergence is interpreted as the uniform convergence on compact sets (u.o.c.).

Uniform convergence on compact sets, as it pertains to fluid limits of networks, is discussed, for instance, in Chen and Mandelbaum [8].

The existence of fluid limits is well-known. A standard argument like the one in Dai [12] shows that for any $r \to \infty$ and any sample path ω, there is a subsequence r_n such that $\bar{S}^{r_n}(\cdot,\omega)$ and $\bar{T}^{r_n}(\cdot,\omega)$ converge as $n \to \infty$. Fix an ω that satisfies (7.1). The convergence of $\bar{T}^{r_n}$, together with equation (7.23), condition (7.1) and Lemma 7.19, implies that $\bar{D}^{r_n}$ converges. This latter convergence, together with equation (7.15) and condition (7.1), implies that $\bar{A}^{r_n}$ converges. The convergence of other components of $\bar{\mathbb{X}}^{r_n}$ then readily follows. Thus, $\bar{\mathbb{X}}^{r_n}$ converges to a fluid limit as $n \to \infty$.

We now convert the fluid limit $\bar{\mathbb{X}}$ into an artificial "fluid limit" $\breve{\mathbb{X}}$. We use the term "limit" facetiously. In fact, the process $\breve{\mathbb{X}}$ is not a limit at all. Our true intentions are embodied in the following proposition. By the expression $\breve{T} = \bar{S} + \bar{T}$ we mean $\breve{T}(t) = \bar{S}(t) + \bar{T}(t)$ for each $t \geq 0$.

Proposition 7.4 *Given a fluid limit* $\bar{\mathbb{X}} = (\bar{A}, \bar{D}, \bar{S}, \bar{T}, \bar{U}, \bar{Y}, \bar{Z})$ *of a processing network operating under a sensible production policy* π*, the process* $\breve{\mathbb{X}} = (\breve{A}, \breve{D}, \breve{T}, \breve{U}, \breve{Y}, \breve{Z}) = (\bar{A}, \bar{D}, \bar{T} + \bar{S}, \bar{U}, \bar{Y}, \bar{Z})$ *is an artificial fluid model solution.*

Proof. Fix the fluid limit $\bar{\mathbb{X}}$ and construct $\breve{\mathbb{X}}$ by collapsing the allocation processes $\bar{S}$ and $\bar{T}$ to $\breve{T}$, i.e., $\breve{T} = \bar{S} + \bar{T}$. Equation (7.41) follows from (7.23), Lemma 7.19 and the fact that $\breve{T}_k(t_2) - \breve{T}_k(t_1) \geq \bar{T}_k(t_2) - \bar{T}_k(t_1)$ for every $0 \leq t_1 \leq t_2$. As for equation (7.42), from the strong law of large numbers (7.3), for each $k = 1, \ldots, K$ and $0 \leq t_1 < t_2$,

$$\lim_{r_n \to \infty} \sum_{k' \in \mathcal{C}(j)} \left(\bar{\Upsilon}^{r_n}_{k'k}(\bar{S}^{r_n}_{k'k}(t_2)) - \bar{\Upsilon}^{r_n}_{k'k}(\bar{S}^{r_n}_{k'k}(t_1)) \right) \geq \frac{1}{s_k} \left(\bar{S}_k(t_2) - \bar{S}_k(t_1) \right), \tag{7.45}$$

and, from (7.22), (7.23) and Lemma 7.19,

$$\bar{D}_k(t_2) - \bar{D}_k(t_1) = B_k \mu_k \left(\bar{T}_k(t_2) - \bar{T}_k(t_1) \right), \quad \text{for each } k \in \mathcal{C}(j), \tag{7.46}$$

whenever $\bar{U}_j(s) > 0$ for each $s \in [t_1, t_2]$. By (7.24), (7.45) and (7.46) we have,

$$\bar{S}_k(t_2) - \bar{S}_k(t_1) \leq \frac{s_k}{\ell_k m_k} \left(\bar{T}_k(t_2) - \bar{T}_k(t_1) \right),$$

and, hence, by (7.11)

$$\begin{aligned} \breve{T}_k(t_2) - \breve{T}_k(t_1) &= \bar{T}_k(t_2) + \bar{S}_k(t_2) - \left(\bar{T}_k(t_1) + \bar{S}_k(t_1) \right) \\ &\leq \frac{\breve{m}_k}{m_k} \left(\bar{T}_k(t_2) - \bar{T}_k(t_1) \right), \end{aligned} \tag{7.47}$$

whenever $\bar{U}_j(s) = \breve{U}_j(s) > 0$ for each $s \in [t_1, t_2]$. Equation (7.42) follows from (7.46) and (7.47). Other fluid model equations can be verified as in Dai [12]. □

By now it should be abundantly clear that artificial fluid models are the combination of a law-of-large-numbers limiting procedure, the limitations on setup allocation via sensible policy constraints, and the collapsing of setup and service allocation processes into a single artificial server allocation process.

Theorem 7.5 *Let a sensible production policy* π *be fixed. If the artificial fluid model is weakly stable, then the corresponding processing network is rate stable.*

Proof. The theorem was first explicitly stated in Chen [7] for standard networks. The only difference here is recognizing that a fluid limit of the processing network is a solution to the artificial fluid model once the allocation processes are collapsed. The remainder of the proof is identical to one for the standard network. See, for example, Dai [13]. □

7.4 The connection between the artificial and standard fluid models

Our goal is to craft a sensible production policy π such that the processing network operating under the policy π is rate stable, if at all possible. As stated earlier, this pursuit is possible only if the usual traffic condition (7.6) holds. It turns out that when (7.6) holds, coming up with a stabilizing π is always possible.

The standard fluid model has become the conventional tool for demonstrating stability of the standard network operating under a given dispatch policy. The connection between stability of the standard network and its associated fluid model was made concrete through the standard network analog to Theorem 7.5, provided by Chen [7]. A sizeable body of literature is devoted to investigating standard fluid models, primarily in demonstrating some form of stability. One particularly useful technique for demonstrating stability is via Lyapunov functions. Not surprisingly, a large portion of the literature focuses on finding the right Lyapunov function to demonstrate stability, given the dispatch policy in question.

Given the similarities between the standard fluid model and the artificial fluid model, we would like to piggyback on these efforts. In particular, it would be ideal if the Lyapunov function that demonstrates the stability of a standard fluid model also works for the artificial fluid model analog. In this section we explore the connections between the fluid models. We start with the special case of batch processing networks, where setups times are all zero, and conclude with the general case.

7.4.1 Batch processing networks and normal policies

A batch processing network is the special case of a processing network for which the setup times are all zero. This definition is slightly more general than the one in Dai and Li [17]. In their paper, each station is populated by a single server. Let $\pi = (\theta, \tilde{\pi}, \ell)$ be a sensible production policy used to schedule a batch processing network. Since setup times are zero, we assume that $\ell_k = 1$ for each class k. Borrowing from Dai and Li [17], the resulting sensible production policy is referred to as a *full batch policy*. A dispatch policy $\tilde{\pi}$ is said to *induce* a full batch policy π if $\pi = (\theta, \tilde{\pi}, 1)$. We now consider when dispatch policies that are known to stabilize standard networks induce full batch policies that stabilize batch processing networks.

Let $\bar{\mathbb{X}} = (\bar{A}, \bar{D}, \bar{S}, \bar{T}, \bar{U}, \bar{Y}, \bar{Z})$ be a fluid limit of a batch processing network. We include the component $\bar{S}$, even though it is always zero, to emphasize that this is still the fluid limit of a processing network. We can trivially construct a process $\breve{\mathbb{X}} = (\breve{A}, \breve{D}, \breve{T}, \breve{U}, \breve{Y}, \breve{Z})$ by simply copying all of the nontrivial components of $\bar{\mathbb{X}}$; that is $(\breve{A}, \breve{D}, \breve{T}, \breve{U}, \breve{Y}, \breve{Z}) = (\bar{A}, \bar{D}, \bar{T}, \bar{U}, \bar{Y}, \bar{Z})$. We refer to processes $\breve{\mathbb{X}}$ constructed in this manner as batch fluid

limits. The batch fluid model then is defined by (7.35)-(7.41), along with

$$\breve{D}_k(t_2) - \breve{D}_k(t_1) = B_k \mu_k \left(\breve{T}_k(t_2) - \breve{T}_k(t_1) \right) \quad \text{if } \breve{U}_j(s) > 0 \quad \forall s \in [t_1, t_2], \quad 0 \le t_1 < t_2 \tag{7.48}$$

and

$$\text{additional equations associated with the particular full batch policy } \pi, \tag{7.49}$$

which replace (7.42) and (7.43), respectively.

In a sense, the batch fluid model bridges the gap between artificial and standard fluid models. For example, as in the artificial fluid model, there are two equations governing the conversion of allocated server effort to departing jobs. In fact, to further match (7.42), equation (7.48) could be expressed as an inequality. Equality holds because the upper bounds and lower bounds agree on intervals during which there is positive fluid at the associated station. Since there are no setup times, the class k inflated mean service times $\breve{m}_k$ (see equation (7.11)) equals the regular class k mean service time m_k. Modulo the server pool constant p_j implicit in the computation of the class k service time in the corresponding standard network $\tilde{m}_k$ (see equation (7.7)), equation (7.48) matches its standard fluid model analog (7.33) on intervals during which $\breve{U}_j(\cdot)$ is positive.

Let $\breve{\mathbb{X}} = (\breve{A}, \breve{D}, \breve{T}, \breve{U}, \breve{Y}, \breve{Z})$ be a batch fluid model solution, i.e., a solution to (7.35)-(7.41), (7.48) and (7.49). We would like to convert it into a standard fluid model solution $\hat{\mathbb{X}} = (\hat{A}, \hat{D}, \hat{T}, \hat{U}, \hat{Y}, \hat{Z})$. We define $\hat{\mathbb{X}}$ as follows: for each $t \ge 0$,

$$\hat{A}(t) = \breve{A}(t), \tag{7.50}$$

$$\hat{D}(t) = \breve{D}(t), \tag{7.51}$$

$$\hat{T}_k(t) = \tilde{m}_k \hat{D}_k(t), \ k = 1, ..., K, \tag{7.52}$$

$$\hat{Y}_j(t) = t - \sum_{k \in \mathcal{C}(j)} \hat{T}_k(t), \quad j = 1, \ldots, J, \tag{7.53}$$

$$\hat{U}(t) = \breve{U}(t), \tag{7.54}$$

$$\hat{Z}(t) = \breve{Z}(t). \tag{7.55}$$

One can verify that the constructed process $\hat{\mathbb{X}}$ satisfies (7.27)-(7.33) such that it is *almost* a standard fluid model solution. As forwarded by Dai and Li [17], we define a subclass of dispatch policies $\tilde{\pi}$ such that the constructed process is indeed a standard fluid model solution.

Definition 7.6 *A dispatch policy $\tilde{\pi}$ is called* normal *if for any batch fluid model solution $\breve{\mathbb{X}}$ under a full batch policy $\pi = (\theta, \tilde{\pi}, 1)$ induced by $\tilde{\pi}$, $\hat{\mathbb{X}}$ constructed by (7.50)-(7.55) also satisfies (7.34).*

For the single-server stations case, Dai and Li [17] show that static buffer priority (SBP) policies, the first-in–first-out (FIFO) policy, and the generalized round robin (GRR) policy are normal. Because we modify dispatch policies in a different manner than in Dai and Li [17], generalized round robin is not normal for our circumstances. It should not be surprising that normal policies for single-server networks are often normal for the multi-server systems.

Proposition 7.7 *If a dispatch policy $\tilde{\pi}$ operating in a standard network is normal, then the batch fluid model under the induced full batch policy π is weakly stable if the standard fluid model under policy $\tilde{\pi}$ is weakly stable.*

Proof. Consider the fluid model of the batch processing network operating under the full batch policy π induced by the normal policy $\tilde{\pi}$. Through a slight modification of the proof by Dai and Li [17], it can be shown that any solution to the batch fluid model $\breve{\mathbb{X}}$ can be converted to $\hat{\mathbb{X}}$ through equations (7.50)-(7.55), where $\hat{\mathbb{X}}$ is a solution to the standard fluid model associated with the dispatch policy $\tilde{\pi}$. □

With this preparation, we now provide an extension of the main result of Dai and Li [17] to networks with multiple servers. Application of the following theorem can be used in conjunction with Corollary 7.15 of Section 7.5.1.

Theorem 7.8 *For a given batch processing network, assume that a dispatch policy $\tilde{\pi}$ is normal for the corresponding standard network. The batch processing network operating under the induced batch policy π is rate stable if the standard fluid model operating under $\tilde{\pi}$ is weakly stable.*

Proof. Assume that $\tilde{\pi}$ is a normal dispatch policy in the corresponding standard network. Assume further that the corresponding standard fluid model is weakly stable. By Proposition 7.7, the batch fluid model operating under the induced batch policy π is weakly stable. The proof follows from Theorem 7.5. □

We conclude with an extension of a result by Dai and Li [17] for so-called *static buffer priority (SBP) policies*. Under an SBP policy, each station has a strict priority ranking of the constituent buffers. When a dispatching decision is made, the associated server is dispatched to the highest priority, eligible buffer. When the network under consideration is a standard one, eligible is synonymous with nonempty. The aforementioned extension can easily be shown:

Lemma 7.9 *Any SBP policy is normal.*

This lemma is critical to Corollary 7.15 at the conclusion of Section 7.5.1.

7.4.2 Stability under sensible production policies

The presence of setups in the processing network renders the arguments of the previous section difficult to apply. That is, extending the definition of normal policies would not be fruitful under our definition of production policy (see the extensions section for discussion of when the extension might be useful). The problem is that, for the artificial fluid model, conversion of allocated server effort to the departure process (embodied in equations (7.41) and (7.42)) is not constant, as it is for a batch fluid model when the associated station has positive fluid (embodied in equation (7.48). Accordingly, we require a more hands-on approach.

Typically, Lyapunov functions are used to demonstrate stability of standard fluid models. The following is one method of using a Lyapunov function. For more examples of its usage, see Dai [13]. Given a standard fluid model solution $\hat{\mathbb{X}} \in \mathbb{D}^{4K+2J}[0,\infty)$:

- Find a functional L which maps $\hat{\mathbb{X}}$ to a nonnegative function f, where $f(t) = L(\hat{\mathbb{X}})(t)$ is absolutely continuous in t, and
- $f(t) > 0$ if and only if $\hat{Z}(t) \neq 0$, such that
- there exists an $\epsilon > 0$ where $\hat{Z}(t) \neq 0$ implies $\frac{d}{dt} f(t) \leq -\epsilon$.

Given such a functional L with the absolute continuity of f, one has $\hat{Z}(t) = 0$ for all $t \geq f(0)/\epsilon$.

This approach is somewhat overkill. The approach allows one to show that, given some positive fluid at time zero, the fluid model solution drains in a finite time proportional to the Lyapunov function value at time zero. But since our fluid model starts with zero fluid, the Lyaponuv function has an initial value of zero, and the fluid model solution remains as such for all time. There is no overkill if the ϵ in the description is equal to zero.

Clearly, Lyapunov functions can also be used for showing weak stability of artificial fluid models. In fact, in some instances:

The Lyapunov function that shows a given dispatch policy $\tilde{\pi}$ stabilizes a standard network may also show that a corresponding sensible production policy π stabilizes a processing network.

This admittedly nebulous statement is the closest we can come to replicating, for general processing networks, the normal policy paradigm for batch fluid models. However, in the following section, we carry out the details in two examples.

The idea just articulated was previously forwarded in Dai and Jennings [15] for systems with no batch processing operations and single-server stations, also referred to as queueing networks with setups. In [15] they examine so-called Kelly networks with setups. A Kelly network, as described

by Bramson [3], is a special class of a standard network, where, for each station, the mean processing times of all constituent classes are identical; that is, $m_k = m_{k'}$ for all $k, k' \in \mathcal{C}(j)$, $j = 1, \ldots, J$. A Kelly network with setups, introduced in [15], has the same mean processing time restrictions of Kelly networks, but the mean setup times have the general sequence-dependent structure of processing networks. Adapting the entropy Lyapunov function used by Bramson [3] to prove stability of Kelly networks, two results emerge: the stability of almost-Kelly networks under the FIFO dispatch policy and the stability of Kelly networks with setups operating under a subclass of sensible FIFO production policies $\pi = (\theta, \text{FIFO}, \ell)$.

Paralleling the result in [15] is the analogous result by Dai and Li [17] for so-called batch Kelly networks, whose predictable definition we omit. It turns out that the FIFO dispatch policy for Kelly networks is a normal policy. Hence, by Theorem 7.8, the batch Kelly network, operating under an induce full batch FIFO policy $\pi = (\theta, \text{FIFO}, 1)$, is rate stable. It would not be surprising if the results in [15] and [17] could be combined for would-be Kelly processing networks.

7.5 Examples of stable policies

We now demonstrate the power of our framework by proving the stability of two sensible production policies. In the first case we consider a special class of multi-type networks operated under a sensible early-steps-first (ESF) policy. In the second case, the processing network with the most general form discussed in the chapter is operated under a sensible generalized round robin (GRR) production policy.

7.5.1 Early steps first

Consider a family of standard networks where each job follows some deterministic *route*, or sequence of buffers, through the network. Suppose that for each buffer there is only one manner in which jobs arrive, exogenously or from one (upstream) buffer. Buffers that receive jobs exogenously are referred to as *sources*. Let $\mathcal{E}$ denote the set of all source buffers. Without loss of generality, $\alpha_k > 0$ for each $k \in \mathcal{E}$. Jobs that enter the system through buffer k are said to belong to the same job *type* and follow the same route of buffers through the network. We refer to such networks as *multi-type queueing networks*, or multi-type networks for short. Jobs of different types do not mix; that is, there are no common buffers in the routes of two distinct job types. In a sense, multi-type networks are the superposition of multiple reentrant lines on a single collection of stations.

For a multi-type network, the corresponding transition matrix P and arrival rate vector α exhibit special structure. If some element k of α is

nonzero, then the corresponding column of P has all zeros. Otherwise, $\alpha_k = 0$ and exactly one element of the corresponding column of P is one and the other elements are all zero.

To facilitate the description of the ESF dispatch policy we alter the notation slightly. In the new notation, classes are denoted by their (type, step) pair. The types are indexed $q = 1, \ldots, |\mathcal{E}|$ and the steps are indexed $k = 1, \ldots, K_q$, where K_q reports the length of the type q route. All of the class-specific quantities now have this alternative notation. For instance, $m_{(q,k)}$ denotes the class (q, k) mean processing time and $Z_{(q,k)}(t)$ records the number of class (q, k) jobs at time t. In a slight abuse of notation, we replace the arrival rate of type q jobs $\alpha_{(q,1)}$ with the quantity α_q.

We are now equipped to describe the ESF dispatch policy. Suppose the server at station j needs to be dispatched at time t. Among all of the nonempty buffers at the station, the server will be dispatched to a buffer (q, k), where k is the first nonempty step at the station. Any other constituent buffer $(q', k') \in \mathcal{C}(j)$ with $k' < k$, must therefore be empty. Note, it is possible that station j houses multiple classes that are the kth step for their respective routes. The policy does not explicitly say how to choose among such classes; that is, ties are broken arbitrarily.

Theorem 7.10 *Under the usual traffic condition (7.6), a multi-type network, operating under the earlier-step-first dispatch policy, is rate stable.*

A proof of Theorem 7.10 might use the following linear Lyapunov function:

$$L(t) = \sum_{(q,k)} \beta^+_{(q,k)} \hat{Z}_{(q,k)}(t), \tag{7.56}$$

where

$$\beta_{(q,k)} = \begin{cases} m_{(q,k)} \left(1 + \frac{\gamma_{j,k}}{1-\rho_j}\right) & k = 1, \ldots, K_q, \\ 0 & k = K_q + 1, \end{cases} \tag{7.57}$$

$$\gamma_{j,k} = \sum_{(q,k) \in \mathcal{C}(j,k)} \alpha_q \beta^+_{(q,k+1)}, \tag{7.58}$$

and

$$\beta^+_{(q,k)} = \sum_{k'=k}^{K_q} \beta_{(q,k')}. \tag{7.59}$$

The set $\mathcal{C}(j, k) = \{(q, k') \in \mathcal{C}(j) : k' = k\}$, used in (7.58), contains the constituent classes of station j that are composed of the kth step of some route. The proof of Theorem 7.10 is implied by the proof of Theorem 7.11.

Multi-type processing networks have the same routing and arrival characteristics as their multi-type network analogs. We can adapt the ESF dispatch policy to form a sensible early-steps-first production policy $\pi = (\theta, \tilde{\pi}, \ell)$. The policy evolves as follows: When a server at station j is free

for dispatching, we create a list of eligible constituent buffers. The server will be dispatched to a buffer (q,k), where k is the earliest step with an eligible buffer at the station. Any other constituent buffer $(q',k') \in \mathcal{C}(j)$ with $k' < k$, must be ineligible. Moreover, if (q',k') is ineligible, it must be the case that

$$Z_{(q',k')}(t) < \theta_{(q',k')} + (p_j - 1)\ell_{(q',k')}B_{(q',k')}.$$

Assuming (q,k) passed the more stringent eligibility test, a setup is performed (if necessary) for class (q,k) and then $\ell_{(q,k)}$ full batches are processed in a row (if possible) before the server is freed for subsequent dispatching. As in the standard network setting, there may be more than one eligible buffer from step k at station j. Any arbitrarily chosen tie-breaking scheme will yield the same stability results.

Theorem 7.11 *A multi-type processing network operating under a sensible early-steps-first production policy $\pi = (\theta, ESF, \ell)$ is rate stable.*

We delay the proof of the following lemma until the appendix.

Lemma 7.12 *Under the sensible early-steps-first production policy $\pi = (\theta, ESF, \ell)$, the artificial fluid model equation (7.43) takes the form*

$$\sum_{k=1}^{k'} \sum_{(q,k)\in\mathcal{C}(j)} \breve{Z}_{(q,k)}(t) > 0 \quad \textit{implies} \quad \sum_{k=1}^{k'} \sum_{(q,k)\in\mathcal{C}(j)} \dot{\breve{T}}_{(q,k)}(t) = p_j \tag{7.60}$$

for every step $k' \geq 1$ and each station $j = 1, \ldots, J$.

Proof of Theorem 7.11. Let $\breve{\mathbb{X}}$ be a solution to the artificial fluid model operating under a sensible ESF production policy. We adapt the linear Lyapunov function devised for the ESF dispatch policy to obtain:

$$L(t) = \sum_{(q,k)} \beta^+_{(q,k)} \breve{Z}_{(q,k)}(t), \tag{7.61}$$

where

$$\beta_{q,k} = \begin{cases} \frac{\breve{m}_{(q,k)}}{B_{(q,k)}} \left(1 + \frac{\gamma_{j,k}}{p_j(1-\breve{\rho}_j)}\right) & k = 1, \ldots, K_q, \\ 0 & k = K_q + 1, \end{cases} \tag{7.62}$$

$$\gamma_{j,k} = \sum_{(q,k)\in\mathcal{C}(j,k)} \alpha_q \beta^+_{(q,k+1)}, \tag{7.63}$$

and

$$\beta^+_{(q,k)} = \sum_{k'=k}^{K_q} \beta_{(q,k')}. \tag{7.64}$$

Notice that equations (7.63) and (7.64) are identical to (7.58) and (7.59), respectively.

Assume that $\breve{Z}(t) \neq 0$ and $\breve{\mathbb{X}}$ is differentiable at time t. Let k be the first system-wide, nonempty step. That is, there is some class (q,k) such that $\dot{\breve{Z}}_{(q,k)}(t) > 0$ and $\dot{\breve{Z}}_{(q,k')}(t) = 0$ for all classes (q,k') such that $k' < k$. Fix step k and time t for the remainder of the proof. We investigate the derivative of the Lyapunov function in (7.61):

$$\begin{aligned}
\dot{L}(t) &= \sum_{(q',k')} \beta^+_{(q',k')} \dot{\breve{Z}}_{(q',k')}(t) = \sum_{q'=1}^{|\mathcal{E}|} \sum_{k'=1}^{K_{q'}} \beta^+_{(q',k')} \dot{\breve{Z}}_{(q',k')}(t) \\
&= \sum_{q'=1}^{|\mathcal{E}|} \sum_{k'=1}^{K_{q'}} \beta^+_{(q',k')} \left(\dot{\breve{D}}_{(q',k'-1)}(t) - \dot{\breve{D}}_{(q',k')}(t) \right) \\
&= \sum_{q'=1}^{|\mathcal{E}|} \left[\alpha_{q'} \beta^+_{(q',1)} - \sum_{k'=1}^{K_{q'}} \left(\beta^+_{(q',k')} - \beta^+_{(q',k'+1)} \right) \dot{\breve{D}}_{(q',k')}(t) \right] \\
&= \sum_{q'=1}^{|\mathcal{E}|} \left[\alpha_{q'} \beta^+_{(q',1)} - \sum_{k'=1}^{K_{q'}} \beta_{(q',k')} \dot{\breve{D}}_{(q',k')}(t), \right]
\end{aligned}$$

where $\breve{D}_{(q',0)}(t) \equiv \alpha_{q'} t$. By the nonnegativity of $\breve{Z}$, $\breve{Z}_{(q',k')}(t) = 0$ implies $\dot{\breve{Z}}_{(q',k')}(t) = 0$. Since k is the first nonempty step, $\dot{\breve{D}}_{(q',k')}(t) = \alpha_{q'}$ for all classes (q',k') such that $k' < k$ and the derivative of the $L(t)$ equals

$$\begin{aligned}
\dot{L}(t) &= \sum_{q'=1}^{|\mathcal{E}|} \left[\alpha_{q'} \beta^+_{(q',k)} - \sum_{k'=k}^{K_{q'}} \beta_{(q',k')} \dot{\breve{D}}_{(q',k')}(t) \right] \qquad (7.65) \\
&\leq \sum_{j:\mathcal{C}(j,k)\neq\emptyset} \left[\sum_{(q,k)\in\mathcal{C}(j,k)} \alpha_q \beta^+_{(q,k)} - \sum_{(q,k)\in\mathcal{C}(j,k)} \beta_{(q,k)} \dot{\breve{D}}_{(q,k)}(t) \right],
\end{aligned}$$

where, again, $\mathcal{C}(j,k) = \{(q,k') \in \mathcal{C}(j) : k' = k\}$ denotes the constituent classes of station j that are composed of the kth step of some route. The transition embodied in (7.65) has two subtleties. For one, each class (q',k') with step $k' \geq k$ the quantity $\dot{\breve{D}}_{(q',k')}(t)$ appears on the left hand side of the inequality whereas the quantity appears on the right hand side for only those classes with step k. Secondly, each nonzero $\alpha_{q'}\beta^+_{q',k'}$ term on the left hand side of the inequality appears on the right hand side as well. The terms that are equal to zero correspond to job types with strictly fewer than k steps. The nonzero values correspond to routes that have at least k steps. Moreover, the kth step must occur at one of the stations. Hence, on the right hand side we can exclude those stations without a resident class that is the kth step of some route. By (7.42), Lemma 7.12 and the fact that

$\dot{\breve{D}}_{(q,k')}(t) = \alpha_q$ if $k' < K$, we have

$$\begin{aligned} \sum_{(q,k)\in\mathcal{C}(j,k)} \dot{\breve{T}}_{(q,k)}(t) &= p_j - \sum_{k'=1}^{k-1} \sum_{(q,k')\in\mathcal{C}(j,k')} \dot{\breve{T}}_{(q,k')}(t) \qquad (7.66) \\ &\geq p_j - \sum_{k'=1}^{k-1} \sum_{(q,k')\in\mathcal{C}(j,k')} \frac{\alpha_q \breve{m}_{(q,k')}}{B_{(q,k')}}. \end{aligned}$$

Hence, by (7.42), (7.64) and (7.65),

$$\begin{aligned} \dot{L}(t) \leq \sum_{j:\mathcal{C}(j,k)\neq\emptyset} \Bigg[& \sum_{(q,k)\in\mathcal{C}(j,k)} \alpha_q \beta_{(q,k)} + \sum_{(q,k)\in\mathcal{C}(j,k)} \alpha_q \beta^+_{(q,k+1)} \\ & - \sum_{(q,k)\in\mathcal{C}(j,k)} \beta_{(q,k)} B_{(q,k)} \breve{\mu}_{(q,k)} \dot{\breve{T}}_{(q,k)}(t) \Bigg]. \end{aligned}$$

It follows from (7.13), (7.62), (7.63) and (7.66) that

$$\begin{aligned} \dot{L}(t) &\leq \sum_{j:\mathcal{C}(j,k)\neq\emptyset} \Bigg[\sum_{(q,k)\in\mathcal{C}(j,k)} \frac{\alpha_q \breve{m}_{(q,k)}}{B_{(q,k)}} \left(1 + \frac{\gamma_{j,k}}{p_j(1-\breve{\rho}_j)}\right) + \gamma_{j,k} \\ &\quad - \left(1 + \frac{\gamma_{j,k}}{p_j(1-\breve{\rho}_j)}\right) \left(p_j - \sum_{k'=1}^{k-1} \sum_{(q,k')\in\mathcal{C}(j,k')} \frac{\alpha_q \breve{m}_{(q,k')}}{B_{(q,k')}}\right) \Bigg] \\ &= \sum_{j:\mathcal{C}(j,k)\neq\emptyset} \Bigg[\gamma_{j,k} - \left(1 + \frac{\gamma_{j,k}}{p_j(1-\breve{\rho}_j)}\right) \\ &\qquad \cdot \left(p_j - \sum_{k'=1}^{k} \sum_{(q,k')\in\mathcal{C}(j,k')} \frac{\alpha_q \breve{m}_{(q,k')}}{B_{(q,k')}}\right) \Bigg] \\ &\leq \sum_{j:\mathcal{C}(j,k)\neq\emptyset} -p_j(1-\breve{\rho}_j). \end{aligned}$$

The derivative of the Lyapunov function is negative as long as there is positive fluid. Hence, the artificial fluid model is stable. □

In addition to implying Theorem 7.10 for standard networks, the proof of Theorem 7.11 for general processing networks implies the intermediate result for processing networks with zero setup times:

Theorem 7.13 *A multi-type batch processing network operating under an induced early-steps-first full batch policy $\pi = (\theta, ESF, 1)$ is rate stable.*

Alternatively, suppose we only knew the following result:

Theorem 7.14 *The fluid model associated with the ESF dispatch policy is weakly stable.*

Given only Theorem 7.14, it is not immediately obvious that Theorem 7.11 holds. However, we could still assert Theorem 7.10 by virtue of the standard network equivalent to Theorem 7.5; see Chen [7]. Moreover, we can also prove Theorem 7.13 through the following argument. First notice that the ESF dispatch policy is an example of a static buffer priority policy, as described in the conclusion of Section 7.4.1. Hence, Lemma 7.9 leads to the following:

Corollary 7.15 *ESF is a normal policy.*

Theorem 7.13 follows immediate from Corollary 7.15, Theorem 7.14 and Proposition 7.7.

7.5.2 Generalized round robin

For standard networks, the generalized round robin (GRR) dispatch policy is parameterized by a set of strictly positive reals $\beta = (\beta_k,\ k = 1, \ldots, K)$. When the constants are integers, the policy works as follows: At station j, the server "visits" the constituent buffers in $\mathcal{C}(j)$ in a fixed cyclic order; hence, the name round robin. In polling station literature, the order in which classes are visited is referred to as the *polling table*, a term we adopt as well. When the server visits buffer k, β_k jobs are processed, if possible. Otherwise, the buffer is exhausted and the server moves on to the next buffer. In this sense, the β_k's can be thought of as nominal allocations. The span of time, from the beginning of the visit to the first buffer in the polling table to the completion of the visit to the last buffer in the table, is referred to as a *cycle*.

When the β_k's are not integers, the spirit of the dispatch policy is the same. However, some requisite bookkeeping is in order. Consider the nth cycle of the server at station j. Let $a_k(n)$ denote the integer-valued nominal allocation for each class $k \in \mathcal{C}(j)$ and $b_k(n)$ denote the nominal residual allocation. The quantities are defined recursively:

$$a_k(n+1) = \lfloor b_k(n) + \beta_k \rfloor \tag{7.67}$$

$$b_k(n+1) = b_k(n) + \beta_k - a_k(n+1), \tag{7.68}$$

for $n = 0, 1, \ldots$, where $b_k(0) = 0$ and, as before, $\lfloor x \rfloor$ denotes the integer part of x. When the server visits buffer k for the nth time, it processes $a_k(n)$ jobs, if possible, before moving on to the next buffer.

For any vector of positive constants β, the additional standard fluid model equation (7.34) takes the form

$$\dot{\hat{T}}_k(t) \geq \frac{\beta_k \tilde{m}_k}{\sum_{k' \in \mathcal{C}(j)} \beta_{k'} \tilde{m}_{k'}}, \quad k = 1, \ldots, K, \tag{7.69}$$

for each t such that $\hat{T}_k(t)$ is differentiable and $\hat{Z}_k(t) > 0$, where, as is our convention, $j = \sigma(k)$. The following theorem is well-known; see, for example, Dai [13].

Theorem 7.16 *Under the usual traffic conditions (7.6), a standard network operating under a generalized round robin policy parameterized by β is stable if, for each $k = 1, \ldots, K$,*

$$\frac{\beta_k \tilde{m}_k}{\sum_{k' \in \mathcal{C}(j)} \beta_{k'} \tilde{m}_{k'}} \geq \lambda_k \tilde{m}_k. \tag{7.70}$$

We now describe how the GRR dispatch policy for standard networks is adapted to form the sensible generalized round robin policy for processing networks, denoted by $\pi = (\theta, \mathrm{GRR}(\beta), \ell)$. Consider the p_j servers at station j. As in the single server case, the servers visit the classes in $\mathcal{C}(j)$ in a round robin fashion. In fact, we use the same nominal values computed in (7.67) and (7.68). However, the interpretations of $a_k(n)$ and $b_k(n)$ are slightly different. The nth visit to buffer k can be made by any (and at most) one of the servers, provided the buffer is eligible. (This is not to say that two servers cannot simultaneously process batches from the same class. It does imply that the servers are performing visits of different cycles.) Suppose one of the servers is performing the nth visit to buffer k. If $a_k(n) \geq 1$ and class k is eligible according to the criterion in Section 7.2.3 then a setup for class k is performed. Otherwise the server moves on to the next buffer. The only time a server does not move is when there is an absence of jobs available for processing at the station. Assuming the server has performed a setup for buffer k, $a_k(n)$ determines how many production runs of length ℓ_k the server will perform. During each production run, each batch contains as many jobs as possible. Suppose the last server was assigned to visit class k in cycle n. The next server to complete its assignment will visit the next class according to the polling table. In this sense, each station's servers march through the cycles in unison. (An alternative description is to have each server perform its own cycle, independent of all other servers.)

Theorem 7.17 *A processing network operating under a sensible generalized round robin policy $\pi = (\theta, GRR(\beta), \ell)$ is rate stable if, for each $k = 1, \ldots, K$, $j = \sigma(k)$,*

$$\frac{p_j \beta_k \ell_k B_k}{\sum_{k' \in \mathcal{C}(j)} \beta_{k'} \ell_{k'} \breve{m}_{k'}} \geq \lambda_k. \tag{7.71}$$

It turns out that when $\beta_k > 1$ for each class k, the condition (7.71) can be relaxed. See Section 7.6 for further discussions. The proof of the theorem depends on the additional artificial fluid model constraint (7.43), described in the following lemma.

Lemma 7.18 *Consider a processing network operating under a sensible generalized round robin policy $\pi = (\theta, GRR(\beta), \ell)$. Artificial fluid solutions obey the following:*

$$\dot{\breve{D}}_k(t) \geq \frac{p_j \beta_k \ell_k B_k}{\sum_{k' \in \mathcal{C}(j)} \beta_{k'} \ell_{k'} \breve{m}_{k'}}, \quad k = 1, \ldots, K. \tag{7.72}$$

When the β_k's are strictly positive integers, the intuition behind (7.72) is straightforward. If station j had only one server, the typical cycle length on average would be at most $\sum_{k' \in \mathcal{C}(j)} (s_{k'} + \beta_{k'} \ell_{k'} m_{k'}) \leq \sum_{k' \in \mathcal{C}(j)} \beta_{k'} \ell_{k'} \breve{m}_{k'}$. When there are enough class k jobs present, the average time a server spends processing those batches in a given cycle is $\beta_k \ell_k m_k$. Since p_j servers act in unison, the average fraction of time spent processing class k jobs is at least the ratio

$$p_j \beta_k \ell_k m_k \left[\sum_{k' \in \mathcal{C}(j)} \beta_{k'} \ell_{k'} \breve{m}_{k'} \right]^{-1},$$

leading to the right hand side of (7.72). The formal proof is delayed until the appendix.

Proof of Theorem 7.17. Let $\breve{\mathbb{X}}$ be an artificial fluid model solution. By Lemma 7.18, if (7.71) holds, we have $\dot{\breve{D}}_k(t) \geq \lambda_k$ for any t such that $\breve{Z}_k(t) > 0$ and $\breve{\mathbb{X}}$ is differentiable at t. It follows from a slight modification of Theorem 4 of Bramson [5] that $\breve{Z}(t) = 0$ for $t \geq 0$. Thus, the artificial fluid model is weakly stable. Rate stability of the processing network follows from Theorem 7.5. □

7.6 Extensions

The naming convention for production policies, as described in Section 7.2.3, has the form $\pi = (\theta, \tilde{\pi}, \ell)$, where θ performs the filtering function, $\tilde{\pi}$ is the dispatch policy, and ℓ determines the length of the production run. We further defined a family of "sensible" policies, where (7.10) and (7.13) hold, to which we have heretofore restricted ourselves. Under certain circumstances, either of these restrictions can be eliminated, or at least relaxed, while still achieving stability.

Consider the generalized round robin policies introduced in Section 7.5.2. A benefit of round robin policies is that they ensure every class is visited on a regular (i.e. cyclic) basis. This removes the need for the threshold test provided by θ. The result is that some production runs may be shorter than required by ℓ, even when this can be avoided. Still, the major results of Section 7.5.2, namely Theorem 7.17 and Lemma 7.18, hold. The reason this works is that round robin dispatching is static and provides every

class the benefit of receiving its designated allocation during each cycle. If a class cannot take advantage of its allotment because of too few jobs, other classes can only benefit. The thresholds are intended to prevent more dynamic policies from favoring classes with too few jobs present.

The pitfalls of exhaustive service were the focus of the first simulation example in the introduction. The idea of exhaustive service, that setups should be avoided as much as possible, is not completely misguided. Indeed excessive setups require too much server effort and can lead to instability. Hence, limited policies were presented as a tradeoff. Rather than effectively eliminating setups as with exhaustive service, limit service reduces the effects of setups to acceptable levels. However, this is not to say that exhaustive service cannot be an effective production run policy. From initial simulation studies, there seems to be a connection between when exhaustive service (with or without setups) leads to instability and when static buffer priority policies cause instability. Under certain conditions all static buffer priority policies are stable. The same should be true for exhaustive service. In fact, we propose the following for investigation: If a system (with or without setups) is unstable under exhaustive service, it is unstable under some static buffer priority policy when setup times are zero. The contrapositive provides a guideline for when exhaustive service should be effective: If there are no static buffer priority policies for which the system without setups is unstable, then the system with setups is stable under exhaustive service.

Full batches, as demanded by our sensible production policy rules, may be an unnecessary extreme. Suppose that station j has a significant amount of excess capacity, i.e., $\breve{\rho}_j$ defined in (7.12) is well below 1. In this case, we can relax full batch size B_k to $\breve{B}_k$, where $\breve{B}_k$ is a target batch size favored by management. We can redefine the inflated traffic intensity (larger than the value from (7.12)) and traffic condition for each station j:

$$\breve{\rho}_j = \frac{1}{p_j} \sum_{k \in \mathcal{C}(j)} \lambda_k (\breve{m}_k / \breve{B}_k) < 1.$$

One can also relax the restriction on the threshold (7.10) as well as the eligibility and non-idling conditions. One advantage of this relaxed requirement on batch sizes is that buffers with smaller numbers of jobs do not have to wait so long for the dispatching of servers.

Consider again the generalized round robin policy. In some cases, the choice of the β_k constants can be used to relax the inflated traffic intensity condition. For instance, suppose β_k is equal to 2. Then in each cycle where there is a sufficient number of jobs on hand, the production run for buffer k will have a length of $2\ell_k$. In this sense, the ℓ_k's that satisfy (7.11) may be unnecessarily large. Just as with the less-than-full batches discussed above, we can define a new inflated traffic intensity (7.12), thus relaxing the traffic condition (7.13). In this case, the new values are based on the new inflated

mean processing time,

$$\breve{m}_k = m_k + \frac{s_k}{\ell_k \max(\beta_k, 1)}.$$

Finally, consider the maximum setup time means $s_k = \max_{k'} s_{k'k}$. These constants, used to define inflated service times and, ultimately, sensible policies, are sometimes overly conservative. For instance, there could be a pair of classes (k', k) whose mean setup time $s_{k'k}$ is relatively large (compared to other setup times at that station). If this particular transition from k' to k can be avoided most of the time, there is no reason to include it in the calculation of s_k. To see how this works, consider the generalize round robin policy and its fixed polling table, or order in which classes are visited. This fixed order suggests a natural way to define s_k. That is, s_k is equal to the mean setup time from the class just before k in the order. Of course, if the class before k in the order is ineligible, the transition to class k will have occurred from some other buffer k'. It could be the case that for this alternative predecessor class k' the mean setup time $s_{k'k}$ is relatively large and not accommodated in the calculation of s_k. There is no alarm because this class k' does not directly precede k relatively often.

7.7 Appendix

7.7.1 Departures as a function of server effort

When there is a single server at some station j, the number of class $k \in \mathcal{C}(j)$ batches processed by time t is captured fully by the quantity $\Psi_k(T_k(t))$. However, when there are multiple servers and random processing times, the quantity can either slightly overshoot or dramatically undershoot the total number of processed batches. Batches begin their processing in FIFO order. However, in the presence of multiple servers, the completion of batches is not necessarily FIFO. The reasoning is that the parallel nature of processing afforded by multiple servers allows for batches with short processing times to emerge before those with long processing time, even when the short batches start later. The quantity $\Psi_k^i(t)$ will provide an upper bound on the number of class k batches processed, given t units of work performed collectively by i servers. This section of the appendix is devoted to providing bounds and fluid limits for $\breve{\Psi}_k(t) \equiv \Psi_k^{p_j}(t)$, for each class k; the results are used in Sections 7.3.1 and 7.5.

Before providing the equations governing Ψ^i, the generalization of Ψ, we first provide a motivating example. Consider three systems, each with a single station network: Systems A, B and C have one, two and three servers, respectively. In each of the systems, 120 minutes of work are performed collectively by the servers. Suppose, for each system, the first 12 batch

processing times (in minutes) are $\{10, 25, 10, 80, 10, 10, 90, 5, 5, 5, 5, 40\}$. If there is no idling, how many batches are processed in each system?

For each of the systems, the following chart matches each server with the batches it processes. In this particular example, the total number of batches processed increases as the number of servers increases. This is not always the case. It is possible that the number of processed batches decreases as the number of servers increases.

System	Server 1	Server 2	Server 3	Partial	Completed
A	b1,b2,b3			b4	3
B	b1, b3	b2,b5,b6		b4,b7	5
C	b1	b2	b3,b5,b6,b8,b9	b4,b7	7

Case A. At time 45 the single server completes the processing of the third batch and starts to process batch number 4. This batch requires 80 minutes of processing and will be completed at time 125. However, the clock stops at time 120. Thus, the number of processed batches at time 120 in the single server system is three.

Case B. With two servers, after t time units there will be $2t$ units of combined server effort. Therefore, the simulation clock stops at real time 60, when 120 combined minutes of server effort has taken place.

At time 20, having processed batches 1 and 3, server 1 is ready to start processing batch number 4, which has 80 minutes of processing requirements. Hence, the completion of this batch will occur at time $20+80 = 100$, well past the clock stopping time of 60. Meanwhile, after 25 minutes, server 2 is ready to start processing batch 5, having completed batch 2. Batch 5 has a processing requirement of 10 minutes. This batch will complete its processing at time $25+10 = 35$. Therefore, server 2 still has time to process batch 6 which has a processing requirement of 10. Batch 6 is completed by time $25 + 10 + 10 = 45$ and server 2 starts on batch number 7. With its processing requirement of 90 minutes, batch 7 will not be completed until time 135, well past the stopping time of 60 minutes.

Case C. The simulation clock stops at time $120/3 = 40$ minutes. Server 1 completes batch 1 after 10 minutes and starts batch 4, which finishes at time 90. Server 3 finishes batch 3 at time 10 then moves on to batches 5 which it finishes at time 20. Then server 3 starts to process batch 6 which it will finish at time 30. Meanwhile server 2 completes batch 2 after 25 minutes and then starts batch 7, which will finish at time 115. After batch 6, server 3 picks up batches 8 and then 9, finishing the last of these at time 40.

Consider the first two cases. Under both scenarios, a server gets "stuck" on the batch with the 80 minute processing time. In the second case, server 2 moves past batch 4, before it too gets stuck on a long batch. Typically, as the number of servers increases, more and more batches with atypically

long processing times can be bypassed. Indeed, in the third case, the 80 and 90-minute long processing times still do not completely impede the processing of additional batches. The question remains: How can we obtain a bound on the number of processed batches given the collective server effort and the sequence of processing times?

Let $\Psi^i(t)$ denote, for a system with i servers, the largest integer value n such that the sum of the processing times of the first n batches is less that t. As stated earlier, when there is one server, the quantity $\Psi(120) = \Psi^1(120) = 3$ captures the number of batches processed, given 120 minutes of effort. With two servers, server 1 gets stuck on batch 4, which happens to be the longest batch experienced thus far, and server 2 gets to look further along in the list of processing times. In the most extreme case, server 1 has just started working on batch 4. In this case, the maximum distance that server 2 can "look into the future" is 80 minutes. (Actually the value is 60, since each server works exactly 60 minutes. But typically, the station-level time is not evenly split.) Hence, we have an upper bound when there are two servers $\Psi^2(120) = \Psi^1(200) - 1 = 5$. We subtract 1 from the quantity to avoid needlessly counting the 80-minute batch.

Next we increase the number of servers to three. In this case, the two largest processing times happen to occupy two of the servers at the horizon's end. Once again, in the worst case, those occupied servers just started processing those long batches. Hence, we need to look 170 minutes into the future to see how many jobs were processed to get $\Psi^3(120) = \Psi^1(290) - 2 = 9$. (Again, this is overkill.) We see here that we have overestimated the number of processed batches. Our objective is for an upper bound, not necessarily a tight one. One should note that we discovered the 90-minute batch, that is skipped over in the 3-server system, only after analyzing the 2-server system. In general, one needs to examine the n-server system in order to provide bounds for the $(n+1)$-server system.

We now provide an iterative method for computing an upper bound on the number of processed batches. Let $\Gamma_k^n(t)$ denote the number of processed class k batches by time t, given there are n servers at station $\sigma(k)$. Recall the formal definition of $\Psi_k = (\Psi_k(t), t \geq 0)$, $\Psi_k(t) = \max\{n \geq 0 : V_k(n) \leq t\}$ and that $\Psi_k^1(T_k(t)) = \Psi_k(T_k(t)) = \Gamma_k^1(t)$. In order to define $\breve{\Psi}$ we need to examine the interjump times of V_k. The quantity $\eta_k(n) = V_k(n) - V_k(n-1)$ tracks the amount of server effort required to process the nth class k batch. It follows that $\sum_{i=1}^n \eta_k(i) = V_k(n)$. Recall that $\Psi_k^n(t)$ is an upper bound on the number of processed batches, given t units of collective work. With $\Psi_k^1(t) = \Psi_k(t)$ we iteratively define the following quantities:

$$\eta_k^{i,t} = \max\{\eta_k(n) : 1 \leq n \leq \Psi_k^i(t) + 1\} \tag{7.73}$$

and

$$\Psi_k^{i+1}(t) = \max\{n : V_k(n) \leq t + i\eta_k^{i,t}\} - i, \tag{7.74}$$

for each $i = 1, \ldots, p_j - 1$. Set $\breve{\Psi}_k(t) = \Psi_k^{p_j}(t)$. Notice that $\eta_k^{1,t}$ is the largest

possible batch processing time that could have been interrupted, given t units of work and 1 server. Now consider adding one server. Since the long batch associated with $\eta_k^{1,t}$ may not have been processed, we do not want its processing time to be deducted from the total server effort t expended by the two servers. Hence, we add $\eta_k^{1,t}$ to t to reflect the virtual server effort for two servers. Once we compute a bound for the number of batches with two servers, we find $\eta_k^{2,t}$, the largest possible batch processing time when there are two servers. To increase to three servers, the virtual server effort would be $\eta_k^{1,t}+\eta_k^{2,t}+t$. However, we simplify matters with the larger quantity $2\eta_k^{2,t}+t$. We iterate until we find $\breve{\Psi}_k(t)=\Psi_k^{p_j}(t)$. Since for each i, $\Gamma_k^i(t)\le\Psi_k^i(T_k(t))$, it follows that

$$\Gamma_k^{p_j}(t)\le\breve{\Psi}_k(T_k(t)),\quad t\ge 0. \tag{7.75}$$

So far we have focused exclusively on the fact that extra jobs may get served as a result of the splitting of processing tasks among several servers. It turns out that the splitting of processing effort can lead to fewer jobs being processed. The lower bound on the number of batches processed is quite straightforward. For each additional server, at most one fewer jobs may have been processed by time t. That is, given there are p_j servers performing the work on class k batches:

$$\Gamma_k^{p_j}(t)\ge\Psi_k(T_k(t))-p_j+1. \tag{7.76}$$

We are interested in taking fluid limits of the $\breve{\Psi}_k$ processes and comparing them to fluid limits of the Ψ_k processes. We have the following lemma, which is used in the proof of Proposition 7.4.

Lemma 7.19 *For any finite integer $i\ge 1$,*

$$\lim_{t\to\infty}\frac{\Psi_k^i(t)}{t}=\mu_k,\quad k=1,\dots,K; \tag{7.77}$$

in particular, $\breve{\Psi}_k(t)/t\to\mu_k$.

Proof. By the law of large numbers (7.2) we have $\Psi_k(t)/t\to\mu_k$. It follows that $\eta_k^{1,t}/t\to 0$. Suppose $\eta_k^{i,t}/t\to 0$ as $t\to\infty$ for some arbitrarily chosen integer i. Then

$$\frac{\Psi_k^{i+1}(t)}{t}\le\frac{\Psi_k(t+i\eta_k^{i,t})}{t}=\left(\frac{\Psi_k(t+i\eta_k^{i,t})}{t+i\eta_k^{i,t}}\right)\left(\frac{t+i\eta_k^{i,t}}{t}\right)\to\mu_k.$$

It follows that

$$\frac{\eta_k^{i+1,t}}{t}\le\left(\frac{\eta_k^{1,t+i\eta_k^{i,t}}}{t+i\eta_k^{i,t}}\right)\left(\frac{t+i\eta_k^{i,t}}{t}\right)\to 0.$$

The proof follows from induction. □

7.7.2 Proofs of Lemmas 7.12 and 7.18

Proof of Lemma 7.12. Let $\bar{\mathbb{X}}$ be a fluid limit of a processing network operating under the sensible early-steps-first production policy $\pi = (\theta, \mathrm{ESF}, \ell)$. Let $\breve{\mathbb{X}}$ be the associated artificial fluid limit, as constructed in Proposition 7.4. Let $\omega \in \Omega$ be a sample path on which (7.1) holds and $\{r_n\}$ be a sequence of positive reals such that $r_n \to \infty$ and $\bar{\mathbb{X}}^{r_n}(\cdot, \omega) \to \bar{\mathbb{X}}(\cdot)$ u.o.c. as $n \to \infty$. Fix station j and a time $t > 0$ such that $\bar{\mathbb{X}}(t)$ and $\breve{\mathbb{X}}(t)$ are differentiable. Suppose that, for some fixed step k_0 we have

$$\sum_{k=1}^{k_0} \sum_{(q,k)\in\mathcal{C}(j)} \breve{Z}_{(q,k)}(t) > 0.$$

Then for some class $(q_1, k_1) \in \mathcal{C}(j)$ with $k_1 \le k_0$ we have $\breve{Z}_{(q_1,k_1)}(t) = \bar{Z}_{(q_1,k_1)}(t) > 0$. By the continuity of $\bar{Z}$, and the uniform convergence $\bar{Z}^{r_n}(\cdot) \to \bar{Z}(\cdot)$ as $n \to \infty$, there exists a $\delta > 0$ and an integer N such that, for each $n \ge N$

$$Z_{(q_1,k_1)}(s) \ge \theta_{(q_1,k_1)} + (p_j - 1)\ell_{(q_1,k_1)} B_{(q_1,k_1)} \quad \forall s \in [r_n t, r_n(t+\delta)].$$

This condition ensures that, throughout the interval $[r_n t, r_n(t+\delta)]$, the servers at station j are never dispatched to a class $(q,k) \in \mathcal{C}(j)$ such that $k > k_1$. This does not mean that these classes composed of later steps receive no server effort whatsoever. Any of the p_j servers may have been dispatched to one such class just prior to $r_n t$. So for the combined server pool at station j, the total allocation, during $[r_n t, r_n(t+\delta)]$, to a class $(q,k) \in \mathcal{C}(j)$ with $k > k_1$ does not exceed p_j class (q,k) setups and $p_j \ell_{(q,k)}$ class (q,k) batches. Equivalently, for each class (q,k) with $k > k_1$,

$$T_{(q,k)}(r_n(t+\delta)) - T_{(q,k)}(r_n t) \le V_{(q,k)}(M^n_{(q,k)} + p_j \ell_{(q,k)}) - V_{(q,k)}(M^n_{(q,k)}) \tag{7.78}$$

and

$$\begin{aligned} &S_{(q,k)}(r_n(t+\delta)) - S_{(q,k)}(r_n t) \qquad (7.79)\\ &\quad \le \sum_{(q',k')\in\mathcal{C}(j)} F_{(q',k')(q,k)}(R^n_{(q',k')(q,k)} + p_j) - F_{(q',k')(q,k)}(R^n_{(q',k')(q,k)}), \end{aligned}$$

where $M^n_{(q,k)}$ is the number of class (q,k) batches processed by time $r_n t$ and $R^n_{(q',k')(q,k)}$ is the number of setups from class (q',k') to class (q,k) performed by time $r_n t$. By Lemma 7.20, the law of large numbers (7.1), and (7.78) and (7.79)

$$\begin{aligned} &\sum_{k'>k_1} \sum_{(q',k')\in\mathcal{C}(j)} \left[\breve{T}_{(q',k')}(t+\delta) - \breve{T}_{(q',k')}(t)\right] \\ &\quad = \lim_{n\to\infty} (1/r_n) \sum_{k'>k_1} \sum_{(q',k')\in\mathcal{C}(j)} \big[S_{(q',k')}(r_n(t+\delta)) - S_{(q',k')}(r_n t)) \\ &\qquad + T_{(q',k')}(r_n(t+\delta)) - T_{(q',k')}(r_n t)\big] = 0. \qquad (7.80) \end{aligned}$$

By (7.39), (7.40) and (7.80),

$$\sum_{k'=1}^{k_1} \sum_{(q',k')\in\mathcal{C}(j)} \left[\breve{T}_{(q',k')}(t+\delta) - \breve{T}_{(q',k')}(t)\right] = p_j\delta.$$

We obtain the result by dividing by δ and letting $\delta \downarrow 0$. □

Proof of Lemma 7.18. Let $\bar{\mathbb{X}}$ be a fluid limit of a processing network operating under the sensible generalized round robin production policy $\pi = (\theta, \mathrm{GRR}(\beta), \ell)$. Let $\breve{\mathbb{X}}$ be the associated artificial fluid limit, as constructed in Proposition 7.4. Let $\omega \in \Omega$ be a sample path on which (7.1) holds and $\{r_n\}$ be a sequence of positive reals such that $r_n \to \infty$ and $\bar{\mathbb{X}}^{r_n}(\cdot,\omega) \to \bar{\mathbb{X}}(\cdot)$ as $n \to \infty$. At time t, suppose that for some class k at station j we have $\breve{Z}_k(t) = \bar{Z}_k(t) > 0$. We would like to show that

$$\dot{\breve{D}}_k(t) = \dot{\bar{D}}_k(t) \geq \frac{p_j\beta_k\ell_k B_k}{\sum_{k'\in\mathcal{C}(j)} \beta_{k'}\ell_{k'}\breve{m}_{k'}}. \tag{7.81}$$

By the continuity of $\bar{Z}$ we know there exists an $\epsilon > 0$ and a $\delta > 0$ such that $\bar{Z}_k(s) > \epsilon$ for each $s \in [t, t+\delta]$. For large enough n, $Z_k(s) \geq \theta_k + \lceil\beta_k\rceil p_j\ell_k B_k$ for each $s \in [r_n t, r_n(t+\delta)]$. Hence, if the ith cycle takes place entirely within the interval $[r_n t, r_n(t+\delta)]$, the server that visits class k during that cycle processes exactly $a_k(i)\ell_k$ batches. It should be clear by (7.67) and (7.68) that, for any class k and positive integers p and q,

$$q\beta_k - 1 < \sum_{i=p+1}^{p+q} a_k(i) < q\beta_k + 1. \tag{7.82}$$

For each n, we refer to cycles that start after $r_n t$ and end before $r_n(t+\delta)$ as *complete*. Let N^n denote the number of complete cycles. The result (7.81) will follow if we can show that, on almost every sample path,

$$\lim_{n\to\infty} \frac{D_k(r_n(t+\delta)) - D_k(r_n t)}{N^n} \to \beta_k\ell_k B_k \tag{7.83}$$

and that

$$\limsup_{n\to\infty} \frac{r_n\delta}{N^n} \leq \frac{1}{p_j} \sum_{k'\in\mathcal{C}(j)} \beta_{k'}\ell_{k'}\breve{m}_{k'}. \tag{7.84}$$

Notice that, in addition to the N^n complete cycles, there may be $2p_j$ *incomplete* cycles. That is, all p_j servers may have been in the middle of a cycle at time $r_n t$ and again at time $r_n(t+\delta)$. By (7.82), we can bound the number of jobs processed in $[r_n t, r_n(t+\delta)]$,

$$(N^n\beta_k - 1)\ell_k B_k < D_k(r_n(t+\delta)) - D_k(r_n t) < ((N^n + 2p_j)\beta_k + 1)\ell_k B_k. \tag{7.85}$$

By Lemma 7.21, $N^n \to \infty$. Dividing both sides of (7.85) by N^n and letting $n \to \infty$ yields (7.83).

Now we demonstrate that (7.84) holds. Clearly none of the p_j servers idle during $[r_n t, r_n(t+\delta)]$ so that

$$p_j r_n \delta = \sum_{k' \in \mathcal{C}(j)} \left[(T_{k'}(r_n(t+\delta)) - T_{k'}(r_n t)) + (S_{k'}(r_n(t+\delta)) - S_{k'}(r_n t)) \right].$$

We investigate how the server effort throughout the interval $[r_n t, r_n(t+\delta)]$ is allocated. Let $M^n_{k'}$ denote the number of class k' batches that have completed service at time $r_n t$. In line with the arguments of Appendix 7.7.1, the $M^n_{k'}$ batches that have completed processing may not be the first $M^n_{k'}$ that started processing in $[0, r_n t)$. In fact the $M^n_{k'}$th batch that completed processing by time $r_n t$ could be as high as the $(M^n_{k'} + p_j - 1)$th batch to have started processing before $r_n t$. Because of the parallel processing capabilities of the station j servers, $p_j - 1$ of the class k' batches between batch number 1 and batch number $M^n_{k'} + p_j - 1$ could be still in-process at time $r_n t$. We can bound the total time dedicated to class k'.

$$\begin{aligned} & T_{k'}(r_n(t+\delta)) - T_{k'}(r_n t) \\ & \quad \leq \quad V_{k'}(M^n_{k'} + \lceil (N^n + 2p_j)\beta_{k'} + 1 \rceil \, \ell_{k'}) - V_{k'}(M^n_{k'}) \\ & \qquad + (p_j - 1) \max_{i \leq M^n_{k'} + p_j} \eta_{k'}(i), \end{aligned} \tag{7.86}$$

where, as before, $\eta_{k'}(i) = V_{k'}(i) - V_{k'}(i-1)$. Similarly,

$$\begin{aligned} & S_{k''k'}(r_n(t+\delta)) - S_{k''k'}(r_n t) \\ & \quad \leq \quad F_{k''k'}(R^n_{k''k'} + 2p_j + \lambda^n_{k''k'}) - F_{k''k'}(R^n_{k''k'}) \\ & \qquad + (p_j - 1) \max_{i \leq R^n_{k''k'}} \left(F_{k''k'}(i) - F_{k''k'}(i-1) \right), \end{aligned} \tag{7.87}$$

where $\lambda^n_{k''k'}$ is the number of type (k'', k') setups among the N^n complete cycles and $R^n_{k''k'}$ is the number of completed type (k'', k') setups before time $r_n t$. The reasoning behind (7.86) is as follows. As argued earlier, there are at most $N^n + 2p_j$ cycles, where N^n of them are complete. We can assume the most extreme case, that for each class k' and in each cycle i, exactly $a_{k'}(i)\ell_{k'}$ batches are processed. Using equation (7.82) we bound the total number of processed batches with $((N^n + 2p_j)\beta_{k'} + 1)\ell_{k'}$. For the last term in (7.86), recall that $M^n_{k'} + p_j - 1$ batches may have started processing before time $r_n t$. So far in our discussion of (7.86) we have only accounted for those batches after batch number $M^n_{k'}$. But the (up to) $p_j - 1$ batches interrupted at time $r_n t$ may be any batch between batch numbers 1 through $M^n_{k'} + p_j - 1$. Hence, the last term. As for (7.87), the arguments are similar. It should be clear that $\sum_{k'' \in \mathcal{C}(j)} \lambda_{k''k'} \leq N^n(\max(\beta_{k'}, 1)) + 1$. Dividing both sides of equations (7.86) and (7.87) by N^n yields

$$\limsup_{n \to \infty} \, (1/N^n) \left[T_{k'}(r_n(t+\delta)) - T_{k'}(r_n t) \right] \leq \beta_{k'} \ell_{k'} m_{k'}$$

and

$$\limsup_{n\to\infty} (1/N^n)\left[S_{k'}(r_n(t+\delta)) - S_{k'}(r_n t)\right] \le \beta_{k'} s_{k'}.$$

Finally, by (7.11),

$$\beta_{k'}\ell_{k'}m_{k'} + \beta_{k'}s_{k'} = \beta_{k'}\ell_{k'}\breve{m}_{k'}$$

and, hence, (7.84) follows. □

Lemma 7.20 *Consider any processing network. Let $\{r_n\}$ be a sequence of positive reals such that $r_n \to \infty$ as $n \to \infty$. Let M_k^n be the number of class k batches processed by time $r_n t$ and $R_{k'k}^n$ be the number of type $(k'k)$ setups performed by time $r_n t$. We have, on almost every sample path,*

$$\limsup_{n\to\infty} M_k^n/r_n < \infty$$

and

$$\limsup_{n\to\infty} R_{k'k}^n/r_n < \infty$$

for each $k, k' = 1, \ldots, K$.

Proof. We prove the first result for class k batches. The result for setups is similar. Suppose $\limsup_{n\to\infty} M_k^n/r_n = \infty$. Without loss of generality, $M_k^n/r_n \to \infty$. By definition,

$$V_k(M_k^n) \le r_n t.$$

Dividing both sides by M_k^n and taking the limit yields $m_k \le 0$, a contradiction. □

Lemma 7.21 *Suppose the processing network is operating under a sensible generalized round robin policy. Let $\{r_n\}$ be a sequence of positive reals such that $r_n \to \infty$ as $n \to \infty$. Fix time t, station j and the constant $\delta > 0$. Suppose that, for large enough n, servers at station j never idle during the interval $[r_n t, r_n(t+\delta)]$. Let N^n denote the number of cycles that transpire completely during $[r_n t, r_n(t+\delta)]$. We have*

$$\liminf_{n\to\infty} N^n/r_n > 0 \tag{7.88}$$

Proof. The p_j servers at station j provide $p_j r_n \delta$ units of potential effort during the interval $[r_n t, r_n(t+\delta)]$. All of this effort goes to processing batches or performing setups. Since there are K classes there are, at most, $K^2 - K$ types of setups. This yields a maximum of K^2 activities over which the $p_j r_n \delta$ units of server effort is distributed over the interval. There is a subsequence r_{n_q} with $n_q \to \infty$ as $q \to \infty$ such that one of the activities receives at least $r_{n_q} p_j \delta / K^2$ units of server effort during the interval for each q. Assume class k receives this amount of effort and, without loss of

generality, that this occurs for each number in the original sequence $\{r_n\}$. That is

$$\begin{aligned} V_k(M_k^n + \lceil (N^n + 2p_j)\beta_k + 1 \rceil \ell_k) &- V_k(M_k^n) \\ &+ (p_j - 1) \max_{i \le M_k^n + p_j} \eta_k(i) \ge r_n p_j \delta / K^2, \end{aligned} \tag{7.89}$$

where M_k^n denotes the number of class k batches completed by time $r_n t$. (The case that a setup activity receives this amount of effort can be argued similarly.) The terms in (7.89) are identical to those in (7.86) of the previous proof. If $N^n/r_n \to 0$ then, by Lemma 7.20, dividing both sides of (7.89) by N^n yields $\delta \le 0$. Hence, (7.88) must hold. □

7.8 Notes

There are recent works in the literature that also extend the standard multiclass queueing network model. For instance, Maglaras and Kumar [32], Kumar and Zhang [30], and Dai and Li [17] all treat the batch processing operations issue. An analogous service was performed by Jennings [27] and Andridotter, Ayhan and Down [1] for queueing networks with setups. In [1] the model allows for more flexible servers than in this chapter.

There has been a flurry of research efforts in pinpointing efficient dispatch policies, meaning policies that maximize throughput, given an achievable arrival rate. Examples of policies that work for any network can be found in Bramson [4, 6] and Dai and Li [16]. For policies that stabilize networks with special structure, see Kumar [28], Dai and Weiss [19], Kumar and Kumar [29], Bramson [3] and Chen and Yao [10]. The last of these examples is related to the multi-type network presented in Section 7.5.1.

The method of using Lyapunov functions for demonstrating stability of a fluid model is wide-spread and multifaceted. For example, Bramson [3, 4] uses entropy type Lyapunov functions, Down and Meyn [20] and Dai and Vande Vate [18] advocate piecewise linear Lyapunov functions, and Chen and Zhang [11] forward linear Lyapunov functions.

Acknowledgments

Jim Dai's research is supported in part by NSF grants DMI-9457336 and DMI-9813345 and by TLI-AP, a partnership between National University of Singapore and Georgia Institute of Technology.

7.9 References

[1] Andradóttir, S., Ayhan, H., and Down, D. G., Dynamic server allocation for queueing networks with flexible servers, *Operations Re-*

search, (2001), Submitted for Publication.

[2] Bertsimas, D., Lecture notes on stability of multiclass queueing networks, 1996.

[3] Bramson, M., Convergence to equilibria for fluid models of FIFO queueing networks, *Queueing Systems: Theory and Applications*, **22**, (1996), 5-45.

[4] Bramson, M., Convergence to equilibria for fluid models of head-of-the-line proportional processor sharing queueing networks, *Queueing Systems: Theory and Applications*, **23**, (1997), 1-26.

[5] Bramson, M., Stability of two families of queueing networks and a discussion of fluid limits, *Queueing Systems: Theory and Applications*, **28**, (1998), 7-31.

[6] Bramson, M., Stability of earliest-due-date, first-served queueing networks, *Queueing Systems: Theory and Applications*, (2001), To Appear.

[7] Chen, H., Fluid approximations and stability of multiclass queueing networks I: Work-conserving disciplines, *Annals of Applied Probability*, **5**, (1995), 637-665.

[8] Chen, H. and Mandelbaum, A., Discrete flow networks: Bottlenecks analysis and fluid approximations, *Mathematics of Operations Research*, **16**, (1991), 408-446.

[9] Chen, H. and Shanthikumar, J. G., Fluid limits and diffusion approximations for networks of multi-server queues in heavy traffic, *Discrete Event Dynamic Systems*, **4**, (1994), 269-291.

[10] Chen, H. and Yao, D. D., Stable priority disciplines for multiclass networks, *Lecture Notes in Statistics*, eds by P. Glasserman, K. Sigman and D.D. Yao, Springer-Verlag, (1996), 27-40.

[11] Chen, H. and Zhang, H., Stability of multiclass queueing networks under priority service disciplines, *Operations Research*, **48**, (2000), 26-37.

[12] Dai, J. G., On positive Harris recurrence of multiclass queueing networks: A unified approach via fluid limit models, *Annals of Applied Probability*, **5**, (1995), 49-77.

[13] Dai, J. G., Stability of fluid and stochastic processing networks, *MaPhySto Miscellanea Publication, No. 9*, Centre for Mathematical Physics and Stochastics, 1999.

[14] Dai, J. G., Hasenbein, J., and Vande Vate, J. H., Stability of a three-station fluid network, *Queueing Systems: Theory and Applications*, **33**, (1999), 293-325.

[15] Dai, J. G. and Jennings, O. B., Stabilizing queueing networks with setups, 2001, Preprint.

[16] Dai, J. G. and Li, C., Discrete proportional processor sharing policy for multiclass queueing networks, 20001, Preprint.

[17] Dai, J. G. and Li, C., Stabilizing batch processing networks, *Operations Research*, (2001), To Appear.

[18] Dai, J. G. and VandeVate, J., The stability of two-station multitype fluid networks, *Operations Research*, **48**, (2000), 721-744.

[19] Dai, J. G. and Weiss, G., Stability and instability of fluid models for re-entrant lines, *Mathematics of Operations Research*, **21**, (1996), 115-134.

[20] Down, D. and Meyn, S. P., Piecewise linear test functions for stability and instability of queueing networks, *Queueing Systems: Theory and Applications*, **27**, (1997), 205-226.

[21] El-Taha, M. and Stidham Jr., S., *Sample-Path Analysis of Queueing Systems*, Kluwer, 1999.

[22] Gross, D. and Harris, C. M., *Fundamentals of Queueing Theory*, Wiley, New York, 1985.

[23] Harrison, J. M., Brownian models of queueing networks with heterogeneous customer populations, *Proceedings of the IMA Workshop on Stochastic Differential Systems,* Springer, 1988.

[24] Harrison, J. M., Brownian models of open processing networks: canonical representation of workload, *Annals of Applied Probability*, **10**, (2000), 75-103.

[25] Hasenbein, J. J., Necessary conditions for global stability of multiclass queueing networks, *Operations Research Letters*, **21**, (1997), 87-94.

[26] Jennings, O. B., *Multiclass Queueing Networks with Setup Delays: Stability Analysis and Heavy Traffic Approximation*, Ph.D. dissertation, School of ISyE, Georgia Institute of Technology, 2000.

[27] Jennings, O. B., On the stability of multiclass queueing networks with setups, 2000, Preprint.

[28] Kumar, P. R., Re-entrant lines, *Queueing Systems, Theory and Applications*, **13**, 1993, 87-110.

[29] Kumar, S. and Kumar, P. R., Fluctuation smoothing policies are stable for stochastic reentrant lines, *Discrete Event Dynamical Systems*, **6**, (1996), 361-370.

[30] Kumar, S. and Zhang, H., Stability of reentrant lines with batch servers, 2000, Preprint.

[31] Lu, S. H. and Kumar, P. R., Distributed scheduling based on due dates and buffer priorities, *IEEE Transactions on Automatic Control*, **36**, (1991), 1406-1416.

[32] Maglaras, C. and Kumar, S., Capacity realization in stochastic batch-processing networks using discrete review policies, 1999, Preprint.

[33] Rybko, A. N. and Stolyar, A. L., Ergodicity of stochastic processes describing the operation of open queueing networks, *Problems of Information Transmission*, **28**, (1992), 199-220.

[34] Takagi, H., *Analysis of Polling Systems*, MIT Press, Cambridge, MA, 1986.