Capacity realization in stochastic batch-processing networks using discrete-review policies

Constantinos Maglaras

Information Systems Laboratory
Stanford University
Stanford, CA 94305-9510
Email: maglaras@isl.stanford.edu

Sunil Kumar

Graduate School of Business
Stanford University
Stanford, CA 94305
Email: skumar@leland.stanford.edu

November 1997

Capacity realization in stochastic batch-processing networks using discrete-review policies [1]

Abstract

We present difficulties with realizing the capacity of queueing networks with batch servers under well known scheduling disciplines. We introduce a family of scheduling policies called discrete-review policies, which not only guarantee capacity realization for queueing networks with batch servers, but also contain among them policies that allow us to realize desired behavior on "fluid" scale, including asymptotically optimal policies. In this family of policies system status is reviewed at discrete points in time, and at each such point the controller formulates a processing plan for the next review period, based on the queue length vector observed, which is then executed in open-loop fashion. We outline a proof of capacity realization for every policy in this family, and provide results of simulation studies that establish the usefulness of the proposed family of policies.

[1] *Key words and phrases:* Scheduling, batch processing, re-entrant lines, discrete-review policies, fluid models. *Running title:* Discrete-review policies for batch networks

1 Introduction

In any manufacturing system, one would like the throughput capacity of the entire system to be no smaller than the throughput capacity of each of the individual servers in the system. Throughput capacity of a manufacturing system depends primarily on the capacity of the individual servers in the system, but it also depends on the operating procedures employed, among other things. We say that *capacity realization* has been achieved by a manufacturing system operating under a given set of operating procedures, if it can sustain throughput rates arbitrarily close to the throughput capacity of its bottleneck servers. Of course, it may not be desirable to push throughput rates close to the capacity of the system because of the resulting congestion, but one would like to retain the ability to do so. Alternately, a system that does not realize capacity is unnecessarily wasteful since its performance degrades at much lower throughput rates than a system that does realize capacity. Thus, capacity realization is not about maximizing throughput, but about maximizing the potential to sustain throughput. Capacity realization would be a necessary condition for the adoption of a set of operating procedures.

In this paper, we investigate the problem of capacity realization for manufacturing systems with batch processing servers. These are servers capable of handling more than one job simultaneously, which are common in some manufacturing systems such as semiconductor wafer fabs (well-drive furnaces for example). In general, the throughput rate is a vector, whose components are the throughput rates of each of the job or product types being manufactured in the system. In this paper, we will assume the product mix, that is, the relative proportions of the product types being manufactured is exogenous to the problem under consideration and is fixed beforehand. We will establish our results for all such fixed product mixes satisfying some very mild assumptions.

We consider manufacturing systems operating under two different job release policies or admission control policies: (a) the *Open loop* release policy which admits new jobs into the system at random, independent and identically distributed intervals of time, and (b) the *CONWIP or Closed loop* release policy [30] which maintains the WIP in the system constant and admits a new job into the system only when a job exits the system, having completed all processing. In the case of the open loop policy, the product mix can be maintained by choosing the distributions of the inter-arrival times suitably, and in the case of CONWIP, the product mix can be maintained by determining the type of the job admitted upon the exit of a previous job by sampling from a distribution. We will analyze such manufacturing systems by building an associated queueing network model under the given operating procedure. We then verify that the queueing network model is capable of sustaining the desired level of throughput in order to establish capacity realization in the original manufacturing system.

In the queueing network model of the manufacturing system under the open loop release policy (called an open queueing network), for each scheduling policy u, we can determine a feasible set Λ^u, such that the network is *stable*, or equivalently, it does not have arbitrarily large average queue lengths, when sustaining any throughput $\lambda \in \Lambda^u$. In the queueing network model of the manufacturing system under the CONWIP release policy (called a closed queueing network), we can determine a critical throughput rate λ^* that is the maximum throughput rate that can be

achieved asymptotically as we allow the fixed WIP level in the model to increase without bound, for a fixed product mix. We can thus determine a set Λ^u of all values of λ^* for all possible choices of the product mix, under the scheduling policy u.

In both cases, we say that *capacity realization* has been achieved in the original manufacturing system under the given scheduling policy u, if the set Λ^u in the queueing network model is the largest possible under any scheduling policy. That is, $\Lambda^w \subseteq \Lambda^u$ for any scheduling policy w. For a fixed product mix, the set of achievable throughput rates is determined by the throughput capacity of the bottleneck server in the system. Hence, an interpretation of our definition of capacity realization under a given policy is that the maximum throughput rate achieved by the scheduling policy for a fixed product mix equals the throughput capacity of the bottleneck server under the same mix.

In manufacturing systems without batch processing capability, calculating the throughput capacity of a server is straight forward. One first calculates the total activity time that the server spends in the transformation of each job; this includes subsequent visits to the same server and has the interpretation of the "unit workload". The reciprocal of this number gives the throughput capacity of this server, or in other words the maximal throughput that can be sustained in ideal circumstances. This calculation is then easily extended to the case of multiple job types in a fixed product mix. In manufacturing systems with batch processing servers the throughput capacity of each individual server is calculated by assuming that the server *always* processes full batches, that is, simultaneously processes as many jobs as possible. Thus, in calculating throughput capacity of systems with batch servers, we equate the batch server to a fast server capable of processing one job at a time.

The issue we will address is whether this throughput capacity can be achieved. The motivation is twofold: first, this will describe the "physical" limits of achievable throughput rates of such systems; second, in modeling a manufacturing system with batch servers, one would like to know when it is justified to replace the batch servers by faster single lot servers, or equivalently, when will a set of operating procedures result in efficient utilization of the batch servers of the system. Of course, the primary aim of the system manager is more than efficient utilization of the batch resources. Yet, as we have discussed earlier, one could argue that capacity realization is a prerequisite of any "good" policy.

Recent developments in queueing network analysis have demonstrated that capacity realization even in networks *without batch servers*, is not trivial. This is particularly true of queueing networks with feedback routing such as reentrant lines, which arise in queueing network models of semiconductor fabs. There have been many recent examples of both open networks which are unstable at throughput rates much smaller than the bottleneck capacity [4, 21, 28], as well as closed networks that do not achieve bottleneck throughput as the WIP grows without bound [14, 19]. However, there have been some positive results which have demonstrated that at least some policies can be guaranteed to realize capacity, at least in the case of reentrant lines [6, 8, 18, 19].

It is not at all clear how capacity realization can be achieved in queueing networks with batch servers, which arise as natural models of manufacturing systems with batch processing servers. In fact, as we will demonstrate in the next section, if we apply scheduling policies that are known to realize capacity in queueing network models without batch servers, one need not realize the

capacity of the corresponding queueing network with batch servers. Together with the examples mentioned above, these observations illustrate the fact the realizable capacity in a manufacturing system is a complex issue that depends on virtually every aspect of the system, including the scheduling policy used and the presence of batch servers. The difficulty faced in scheduling policy design for queueing networks with batch servers is in the trade-off between running full batches and incurring excessive delays (and consequently WIP) while waiting for full batches to build up. Ideally we would like the batch servers to always run full batches and yet the network not to incur substantial delays beyond those in an equivalent network composed only of single servers. It is not immediate whether a general procedure to achieve this goal exists. Finally, one of the main approaches to verifying capacity realization for queueing networks involves analyzing their associated "fluid limits" [3, 5, 6, 7, 17, 19]. As it will become apparent in the next section, it is not clear how one can describe fluid limits of networks with batch servers, and thus direct application of this method is difficult.

There is a considerable literature on queueing networks with batch servers. Most of it is either concentrated on performance analysis of single station networks [27, 24], or of product form networks [29, 26] which form a very small class of queueing network models, or on designing batching control policies using simple models [10, 11]. In a different spirit from the work mentioned above, significant contributions have been made in addressing the complexity of scheduling problems with batch servers in a combinatorial optimization framework. The reader is referred to [1] and some of the references therein for some recent results. In all these cases, capacity realization is either obvious or implicitly assumed. In fact, none of the existing literature would lead the reader to suspect the kind of behavior illustrated in the next section.

The main contributions of this paper are the following. First, we will use a simple example in order to illustrate that capacity realization for systems with batch servers can be difficult. The example we use is one where capacity realization is guaranteed under any non-idling scheduling rule in the absence of batching [8]. Thus, it is the presence of batch servers that causes this subtle and undesirable system behavior. Moreover, the analysis of this example and a series of similar counterexamples provided in [20], establish that in general, simple heuristic rules will fail to realize capacity of batch processing systems. This motivates the problem of finding a policy that will guarantee capacity realization for this class of systems. Our second contribution, is to describe a family of scheduling policies, called discrete-review policies, that always achieves capacity realization in queueing networks with batch servers under very general assumptions. These policies were first introduced in Harrison's BIGSTEP approach to dynamic flow management [13] and later, they were formally described and analyzed for a general class of networks by one of the authors in [22, 23]. In a discrete-review policy, system status is reviewed at discrete instants in time and processing decisions are made over a planning horizon whose length depends on the amount of work present in the system upon review. Safety stock requirements are enforced for each class of jobs in order to ensure the proper execution of these processing plans and to avoid unplanned idleness.

Although understanding and solving the problem of capacity realization is fundamental to the design of scheduling policies and must logically precede any other design consideration, this is not an end in itself. Scheduling polices also have to perform well with respect to other (and sometimes

conflicting) metrics like mean throughput time. The family of discrete-review policies allows us to pick policies that not only guarantee capacity realization but also perform well with respect to the other performance metrics. For example, a static priority rule may be desirable from a throughput time standpoint, and yet capacity realization cannot be guaranteed under this priority policy. Using the discrete-review framework, one can construct a policy that is guaranteed to realize capacity, and yet whose gross behavior mimics the static priority policy (by keeping the majority of the WIP in the classes that correspond to the lower priority classes in the static policy, for example). It is also possible to find a discrete-review policy that is "optimal" in an asymptotic fluid sense. Thus, the discrete-review framework goes beyond mere capacity realization. Also, discrete-review policies involve periodic, but relatively infrequent, review and planning. This fits naturally into a practical scheduling system where status is reviewed at the beginning of a shift, and a processing plan is determined for the shift. The operators then implement the plan without supervision, and at the end of the shift, the new status of the manufacturing system is determined and the process repeated. Thus the discrete-review framework is more than a mere theoretical convenience. Finally, the fluid limits are easy to describe under discrete-review policies. This allows us to use the powerful theoretical machinery developed for fluid limits to obtain performance guarantees.

The specific contributions in this paper relative to the previous results of [22, 23] are the following. First, the class of networks under investigation and the corresponding description of the family of discrete-review policies are extended to allow for servers with batch processing capability. All previous results regarding open network models established in [22, 23] are generalized for the case of batch processing systems. Second, closed network models are considered for which it is proved that under fairly general assumptions, every discrete-review policy in this family is guaranteed to realize capacity. The analysis of efficiency of closed networks using fluid models follows on the previous work in [19] and is new for the class of networks and policies considered here. Moreover, from the nature of our results and the associated mode of analysis, it is should be apparent that the proposed framework can be extended to more general network models with relatively simple reasoning, while leaving the overall structure of the proposed solutions unaltered. Such extensions include setup delays (or costs), routing decisions and admission control (job release policies), and more complicated processing paradigms where, for example, multiple resources need to be engaged for one activity. An important implication of this observation from a practical perspective is that more accurate network models of real manufacturing systems can be addressed within this framework.

The rest of the paper is structured as follows. Section 2 describes a simple two-station example, which should help motivate the general problem being addressed. Section 3 describes the open and closed multiclass queueing network models and formally defines capacity realization. In Section 4, a family of discrete-review policies is described both for open network models and in Section 5 an outline of the proof of stability under a a discrete-review policy in this family is presented. Section 6 extends these results to the case of closed queueing networks and Section 7 contains some remarks regarding different choices of such discrete-review policies in relation to other performance criteria of interest. Section 8 contains a simulation study for the example described in Section 2 and finally, some concluding remarks are included in Section 9.

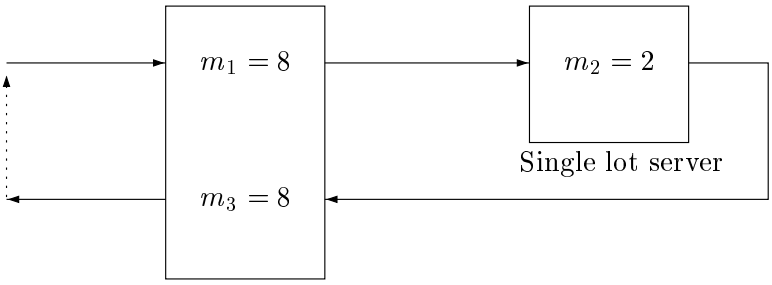

Figure 1: Example of Batch Processing Network

2 Motivating Example

In this section we will present an example of a reentrant line with a batch server. This is a queueing network model which arises as a natural model of process subsequences within semiconductor wafer fabs. This example will serve to highlight the difficulties with capacity realization in manufacturing systems with batch servers.

Consider the reentrant line shown in Figure 1. It consists of two servers, of which server 1 is a batch server with maximum batch size 10, i.e., it is capable of processing up to 10 customers simultaneously. Server 2 has maximum batch size 1. There are three customer classes. Classes 1 and 3 are served by server 1, and class 2 by server 2. New arrivals begin service as class 1 jobs and on completion of service, successively become class 2 and 3 jobs before exiting. The (deterministic) time to serve jobs in classes 1, 2, and 3 is 8, 2, and 8 respectively.

The network is operated under a closed loop or CONWIP admission control scheme. That is, a new job is introduced into class 1 whenever a job departs from class 3. This maintains the number of jobs in the network constant, and this fixed population is set at $N > 30$. The scheduling policy used is the Last Buffer First Serve (LBFS) static priority rule which gives priority to class 3 jobs over class 1 jobs at server 1; that is, higher priority is given to jobs that are closer to completing service in the system and thus exiting. Jobs within a class are served in FIFO. This priority policy is implemented in a non-preemptive manner and both servers 1 and 2 operate in a non-idling fashion. That is, whenever a server has any work to process, it begins processing and does not idle in anticipation of further work buildup. (These policies are also referred to as work conserving.) Also, there is no minimum batch size requirement. That is, server 1 begins working on a batch as soon as there are any jobs in the batch, and does not wait until a minimum number of jobs are available before processing; that is, the server admits $\min(Q_k(t), 10)$ jobs, where for $k = 1, 3$. The system begins operation with all N jobs in class 1.

In order to describe the future dynamics of the system, we will keep track of the number of jobs in each class in the system, the remaining service time of the batch at the head of the line in each of the classes, and the number of jobs in the batch at the head of the line in both class 1 (batch 1) and class 3 (batch 3) at server 1. The (deterministic) evolution of the state, starting from the initial state of N jobs in class 1, is described in Table 1. Table 1 also keeps track of the cumulative

Time	State(Queue; Remaining Service; Batch1, Batch2)	Cum. departures
0	$(N, 0, 0;\ 8, 0, 0;\ 10, 0)$	0
8	$(N-10, 10, 0;\ 8, 2, 0;\ 10, 0)$	0
16	$(N-20, 16, 4;\ 8, 2, 8;\ 10, 4)$	0
24	$(N-16, 12, 4;\ 8, 2, 8;\ 10, 4)$	4
32	$(N-12, 8, 4;\ 8, 2, 8;\ 10, 4)$	8
40	$(N-8, 4, 4;\ 8, 2, 8;\ 10, 4)$	12
48	$(N-4, 0, 4;\ 8, 0, 8;\ 10, 4)$	16
56	$(N, 0, 0;\ 8, 0, 0;\ 10, 0)$	20

Table 1: State Trajectory under LBFS

number of departures from class 3 up to the time indicated.

Table 1 shows that the state trajectory is periodic with period equal to 56. In these 56 time units, the number of departures from class 3 is 20. Thus, the long run average throughput of this system is $5/14 = 0.357$ jobs per unit time. This is smaller than the throughput rate predicted assuming that server 1 always runs full batches of size 10, which is 0.5 jobs per unit time. This result does not depend in the amount of WIP in the system, since one could choose N to be arbitrarily large while the evolution of system dynamics will still be described exactly as in Table 1. Also, it is interesting to note that when we assume that server 1 runs full batches, the bottleneck server is server 2, whereas in the trajectory described in Table 1, the bottleneck server is server 1. That is, despite the fact that the batch server has ample capacity in order to theoretically sustain the bottleneck system throughput, determined by server 2 to be 0.5 jobs per unit time, the simple scheduling rule employed insufficiently utilizes server 1 (by running small batches) so that it becomes the bottleneck resource. An interesting direction of work would be to examine the sensitivity of the overall system throughput on deviations from full batch loading for server 1. The simplest case to be considered would be the extreme one where server 1 is the bottleneck resource to start with, for example in the case where $m_1 = m_3 = 10$, every deviation from operating in full batches would result to a loss in throughput. In general, system throughput sensitivity to batch loading characteristics will depend on the amount of excess capacity for server 1 and the specifics of the implemented policy the system. This is an interesting topic on its own and will not be pursued here any further. Finally, one may be tempted to believe that the behavior described in Table 1 is a consequence of choosing the service times to be deterministic. This is not true: the same conclusion is obtained when the service times are i.i.d. random variables with the same means. Section 7 explores this further.

This example illustrates the following issues.

1. Capacity realization in networks with batch servers is non-trivial, and policies which realize capacity in networks without batch servers may not do so in networks with batch servers. In particular, LBFS is guaranteed to realize capacity in the reentrant line of Figure 1 if we replace server 1 by a server which processes one customer at a time but 10 times faster. Thus,

it may be grossly incorrect to approximate the behavior of networks with batch servers using single, faster server models.

2. Naive policies which do not incorporate minimum batch requirements may not be the desirable class of policies in networks with batch servers. One possible solution is to enforce minimum batch size requirements as in [11]. A minimum batch requirement of 10 at server 1, i.e., not allowing server 1 to process a batch unless it has 10 jobs in it achieves a throughput of 0.5 in this example. But, in general, the choice of the minimum batch size is non-trivial, since the throughput of the system is not monotone in the choice of minimum batch size and sometimes no minimum batch size rule will realize capacity [20]. Also, manufacturing process considerations may determine the minimum batch size rather than operational policies, further complicating the issue.

3. To the reader familiar with fluid models associated with queueing networks, it will be clear that it is very hard to describe the fluid model corresponding to this example in particular, or networks with batch servers in general; this will be discussed further in section 5.

In a subsequent section of this paper we will describe a family of discrete-review policies for networks with batch processing resources. Each policy from this family will guarantee capacity realization and furthermore, it will enforce each server to process jobs in full batches. But first, we formally define the queueing network models we will work with in this paper.

3 Multiclass queueing network models

The results presented in this paper apply both to open and closed queueing network models. For concreteness, we first provide a detailed description of the former, based on the setup introduced by Harrison in [12]. We will then provide a description of how the model may be adapted for the closed network case.

3.1 Open Network Model

Consider a queueing network of single server stations indexed by $i = 1, \ldots, S$. (Throughout this paper the terms station and server will be used interchangeably.) Each station is allowed to process jobs in batches and the maximum batch size at station i will be denoted by c_i. The network is populated by job classes indexed by $k = 1, \ldots, K$ and infinite capacity buffers are associated with each class of jobs. Class k jobs are processed in batches by a unique station $s(k)$ and each station can only process one job class at any point in time. The service times for class k batches are $\{\eta_k(n), n \geq 1\}$, and for concreteness service is assumed to be non-preemptive. Let $\mu_k = 1/\,\mathbf{E}[\eta_k(1)] = 1/m_k$ be the service rate for class k jobs, expressed in batches per unit time, and $M = \mathbf{diag}\{m_1, \ldots, m_K\}$. Upon completion of service at station $s(k)$, a job in a class k batch becomes a job of class m with probability P_{km} and exits the network with probability $1 - \sum_m P_{km}$, independent of all previous history. Assume that the general routing matrix $P = [P_{km}]$ is transient (that is, $I + P + P^2 + \ldots$ is convergent). Every job class k can have its own exogenous arrival process with interarrival times

$\{\xi_k(n), n \geq 1\}$. The set of classes that have a non-null exogenous arrival process will be denoted by $\mathcal{E}$ and the the notation $E(t)$ will be used to signify the K-dimensional vector of exogenous arrivals in the time interval $[0,t]$. For future reference, let $\lambda_k = 1/\mathbf{E}[\xi_k(1)]$ be the arrival rate for class k jobs and $\lambda = (\lambda_1, \ldots, \lambda_K)^T$, and let $M = \mathbf{diag}\{m_1, \ldots, m_K\}$. The set $\{k : s(k) = i\}$ will be denoted C_i and is called the constituency of the server i, while the $S \times K$ constituency matrix C will be the following incidence matrix:

$$C_{ik} = \begin{cases} 1 & \text{if } s(k) = i \\ 0 & \text{otherwise.} \end{cases}$$

Throughout this paper we make the following assumptions on the distributional characteristics of the arrival and service time processes:

(A1) $\xi_1, \ldots, \xi_K$ and $\eta_1, \ldots, \eta_K$ are mutually independent IID sequences;

(A2) $\mathbf{E}[\eta_k(1)] < \infty$ for $k = 1, \ldots, K$ and $\mathbf{E}[\xi_k(1)] < \infty$ for $k \in \mathcal{E}$;

(A3) For any $x > 0$, $k \in \mathcal{E}$, $\mathbf{P}\{\xi_k(1) \geq x\} > 0$. Also, for some positive function $p(x)$ on $\mathbf{R}_+$ with $\int_0^\infty p(x)dx > 0$, and some integer j_0, $\mathbf{P}\left\{\sum_{i=1}^{j_0} \xi_k(i) \in dx\right\} \geq p(x)dx$.

The technical regularity conditions in **(A3)** are imposed so that we can make use of the general stability theory of Dai [6]; these conditions are never invoked in propositions which are actually proved in this paper.

Finally, denote by $Q_k(t)$ the total number of class k jobs in the system at time t, and by $Q(t)$ the corresponding K-vector of "queue lengths". A generic value of $Q(t)$ will be denoted by q, and the size of this vector is defined as $|q| = \sum_k q_k$. Let $\Gamma = \mathbf{diag}\{c_{s(1)}, \ldots, c_{s(k)}\}$ and define $\hat{Q}(t) = \Gamma^{-1}Q(t)$. That is, $\hat{Q}_k(t)$ is the number of class k jobs expressed as a number of full class k batches.

Let $R = (I - P^T)M^{-1}\Gamma$. The *nominal workload* or *average resource utilization level* vector for a multiclass network is defined by

(A4) $$\rho = CR^{-1}\lambda < e,$$

where the inverse exists since P is assumed to be transient. The inequality in condition **(A4)** implies that there is enough processing capacity to deal with incoming traffic, provided that each server is processing work in full batches. (The letter e signifies the vector of ones of appropriate dimension, and all vector inequalities should be interpreted componentwise.)

A scheduling policy takes the form of a K-dimensional cumulative allocation process $T = \{T(t),\ t \geq 0,\ T(0) = 0\}$, where $T_k(t)$ denotes the time allocated by server $s(k)$ into serving class k jobs up to time t. Let $I(t)$ be the I-dimensional cumulative idleness process defined by $I(t) = et - CT(t)$, where $I_i(t)$ is the total time that server i has been idled up to time t. The process $I(t)$ has to be non-decreasing, which reflects the implicit capacity constraint of the system. For purposes of this paper it will not be necessary to define precisely the set of admissible scheduling policies, because our goal is simply to study system behavior under the policies described in the next section.

3.2 Closed Network Model

In order to describe a closed queueing network, one needs to restrict attention to a multiclass network where there are no exogenous arrivals but the switching matrix is assumed to be stochastic. In our notation this translates to the condition that $\lambda_k = 0$ and $\sum_m P_{km} = 1$ for each job class k[2]. Thus, in a closed queueing network initialized by a K-dimensional queue length vector such that $|Q(0)| = N$, these N jobs will circulate indefinitely among stations in the network, with no arrivals and no departures. It is assumed that the switching matrix P is irreducible, and thus jobs of any given class visit all other classes with probability one. The traffic intensity, or nominal load, at every station α_σ can be calculated up to a constant given the switching matrix P and the service rate vector μ. We make the following canonical choice. Let π be the unique invariant probability measure associated with the routing matrix P. This is determined by the transition matrix P and equation $\pi P = \pi$. Then define

$$\alpha = CM\pi, \tag{3.1}$$

and

$$\alpha^* := \min_\sigma \frac{1}{\alpha_\sigma}. \tag{3.2}$$

α^* above is the maximum sustainable throughput at any WIP level. Scheduling policies are defined for closed networks exactly as in the open case.

3.3 Capacity Realization

We now formally define capacity realization in both models described above. In the case of open networks, we say that capacity realization is achieved by a specified scheduling policy if the network is *stable*, i.e, the underlying Markov chain for the network[3] is positive Harris recurrent for *every* λ satisfying **(A4)**; the reader is referred to Dai [6] for a detailed description. Under some more conditions on the arrival and service processes [7], we can define capacity realization by the equivalent condition that the expected total WIP $E|Q(t)| < \infty$ in steady state for *every* λ satisfying **(A4)**. **(A4)** implicitly defines the largest possible set Λ^u described in Section 1. We will never explicitly work with Λ^u, but will use **(A4)** instead.

In the case of closed networks, N denotes the constant WIP level in the network (that is $|Q(t)| = N$), and let $\lambda_k(N)$ be the average departure rate for class k jobs. We say that capacity is realized under a given policy if the policy is "efficient," that is,

$$\lim_{N\to\infty} \lambda_k(N) = \pi_k \alpha^*. \tag{3.3}$$

4 Discrete-review policies

Discrete-review policies were first introduced by Harrison in his BIGSTEP approach to network control problems in [13] and later, they were formally described and analyzed in a general framework

[2]This product mix structure is subsumed in the matrix P since P determines the distribution of the type of a new arrival upon the exit of an old job.

[3]We will be deliberately vague about this underlying Markov process and describe it only for our recommended policy.

by one of the authors in [22] and [23]. In this paper, we extend the work of the later to batch-processing networks.

In such a policy, system status is reviewed at discrete points in time, and at each such point the controller formulates a processing plan for the next review period, based on the queue length vector observed. Formulation of the plan requires solution of a linear program whose objective function involves a dynamic reward function. This function assigns a reward rate to time devoted in processing the different classes of jobs as a function of the observed queue length vector. In our formulation this function can be almost arbitrary, though in practice it can be chosen judiciously in order to "induce" desired system behavior, or enforce the desired priorities among the various job classes. Implementation of the plan involves enforcement of certain safety stock requirements in order to avoid unplanned server idleness. The durations of review periods and the magnitudes of safety stocks are dynamically adjusted: review periods get longer and safety stocks increase as queues lengthen, but both grow less-than-linearly as functions of queue length. During each review period the system is only allowed to process jobs that were present in the beginning of that period, which makes the implementation of the associated processing plans very simple.

In specific, a discrete-review policy is defined by or is derived from a real valued, strictly positive, concave function $l(\cdot)$ on $\mathbf{R}_+$, a a real valued, strictly positive, continuous function $r(\cdot)$ on $\mathbf{R}_+^K$, plus a K-dimensional vector β that satisfy the following restrictions.

$$\frac{l(x)}{\log(x)} > c_0, \quad \frac{l(x)}{\log(x)} \to \infty \ \text{ as } x \to \infty, \ \text{ and} \tag{4.1}$$

$$\frac{l(x)}{x} \to 0 \ \text{ as } x \to \infty, \tag{4.2}$$

$$c_1 \le r_k(q) \le c_2 + |q|^\gamma \quad \text{for some } \ c_1,\ c_2,\ \gamma > 0 \ \text{ and } k = 1, \ldots, K, \tag{4.3}$$

$$\beta_k > \mu_k \ \text{ for } k = 1, \ldots, K. \tag{4.4}$$

Under any of the policies to be considered, system status will be observed at a sequence of times $0 = t_0 < t_1 < t_2 < \ldots$; we call t_r the r^{th} review point and the time interval between t_r and t_{r-1} the r^{th} planning period. Given that the queue length vector $q = Q(t_r)$ is observed at t_r, server activities over the next planning period are determined by solving a linear program, the data for which involve $l(\cdot)$, $r(\cdot)$, and β. We first discuss this linear program for the open network model of section 3.1. To be specific, having observed q the controller sets $\tilde{q} = q/|Q(0)|$,

$$l = l(|\Gamma^{-1} q|), \ r = r(\tilde{q}), \ \text{and } \theta = l\Gamma\beta, \tag{4.5}$$

and then solves the following linear program: choose a K-vector x of time allocations or activity levels to

$$\text{maximize} \quad r^T x \tag{4.6}$$

$$\text{subject to} \quad q + l\lambda - Rx \ge \theta, \quad x \ge 0, \quad Cx \le le. \tag{4.7}$$

(The reader should note that l and r when used without any argument will always correspond to the length of a specific planning period and the corresponding reward vector.) The constraint

$q + l\lambda - Rx \geq \theta$ implies that the nominal ending inventory will be above a specified threshold requirement. It is implicit in this formulation that the planning problem involves a deterministic fluid approximation to system dynamics. It is also clear from the definition of the matrix R as $R = (I - P^T)M^{-1}\Gamma$ that it has been assumed that servers process work in full batches. Assuming for now that this planning linear program is feasible and given the vector of nominal time allocations x, a plan expressed in number of batches of jobs of each class to be processed over the ensuing period ($p(k)$), and a nominal idleness plan expressed in units of time for each server to remain idle (u_i) over the same period are formed as follows:

$$p(k) = \left\lfloor \frac{x_k}{m_k} \right\rfloor \wedge \left\lfloor \frac{q_k}{c_{s(k)}} \right\rfloor \quad \text{for } k = 1, \ldots, K, \quad \text{and} \quad u_i = l - (Cx)_i \quad \text{for } i = 1, \ldots, S. \tag{4.8}$$

To implement this plan, each server i is first idled for u_i units of time, and then the plan p is implemented in open-loop fashion until its completion, which signals the start of the $(r+1)^{th}$ review period. The construction of the processing plan p using equation (4.8) ensures that it will be implementable from jobs present at the beginning of this review period. Condition (4.4) ensures that if the observed state q is above the safety stock requirement θ, then there is enough work at each station in order to process any feasible processing plan in full batches. The actual time allocated in serving jobs during each review period is equal to the execution time of the processing plan p (adjusted for any planned idleness). Following this observation, it should be clear that in general the actual duration of a review period will not be equal to l and thus, a distinction should be made between the nominal and actual allocation processes.

The objective of the linear program (4.6) is defined using the function $r(\cdot)$ that hereafter it will be referred to as a *dynamic reward function*. Such a function associates with each (appropriately normalized) queue length vector $\tilde{q}$ a corresponding K-vector $r(\tilde{q})$, where the k^{th} component $r_k(\tilde{q})$ is treated as a reward rate for time devoted to processing class k jobs. In the planning problem (4.6)-(4.7) one seeks to determine a vector x of time allocations over the planning period that maximize total reward subject to the various constraints imposed. The specific choice of the reward rate function will affect the overall system behavior; pointers as to different such choices will be provided in section 7.

When the planning linear program (4.6)-(4.7) is infeasible, a relaxed - or *infeasible planning* - logic is employed to steer the state above the desired threshold levels. The first step of this infeasible planning algorithm is summarized in the following linear program: find a scalar $\hat{l}$ and a K-vector $\hat{x}$ to

$$\text{minimize} \quad \hat{l} \tag{4.9}$$

$$\text{subject to} \quad \hat{l}\lambda - R\hat{x} > \Gamma\beta + e, \quad \hat{x} \geq 0, \quad \hat{l} \geq 0, \quad C\hat{x} \leq \hat{l}e. \tag{4.10}$$

Given the solution of the linear program (4.9)-(4.10), which is always feasible, a processing plan $\hat{p}(k) = \lfloor \hat{x}_k / m_k \rfloor$ and an idleness budget $\hat{u} = \hat{l}e - C\hat{x}$ are formed. Then the nominal processing plan and idleness budget for the r^{th} planning period is set to be $p_r = \hat{p}J$ and $u_r = \hat{u}J$ respectively, where $J = \lceil l \rceil$. This processing plan cannot be implemented from jobs that are all present at the review point, and as a result a more careful execution methodology should be employed. The specifics of

this algorithm together with a rigorous analysis can be found in [23]. For the extension to networks with batch servers it is necessary to build up the levels to $\Gamma\beta$ and the remaining of the logic in [23] applies without any modification. Note that in simpler network models, such as reentrant lines introduced by Kumar in [15], this infeasible planning step is much easier to implement. There, the system is first idled to accumulate $|\theta|$ external arrivals and subsequently, work is pushed through the network until all job classes are above their safety stock requirement.

Finally, the notation $\mathbf{DR}(r, l, \beta)$ will be used in order to specify a discrete-review policy derived from the functions $r(\cdot)$, $l(\cdot)$, and the vector β. In the sequel, we will use a subscript to differentiate between different review periods.

One can define an underlying continuous time Markov chain for a multiclass network under any policy in the proposed family. At any time t define the S-vector $b(t)$, where $b_i(t)$ is the number of jobs in the current batch being served in station i. Assume that $t_r \leq t < t_{r+1}$ and define the parameter $J(t)$ to be equal to 1 if the linear program (4.6)-(4.7) is feasible or otherwise set it equal to the number of remaining executions of the processing plan $\hat{p}$ derived from (4.9)-(4.10). Let $p(t)$ be a K-vector, where $p_k(t)$ is the number of class k jobs that remain to be processed at time t according to the processing plan p_r or $\hat{p}_r$, depending on whether the planning LP was feasible. Let $u(t)$ be the S-vector of remaining idling times for each of the servers for the ensuing planning period. Finally, let $R_a(t)$ be the $|\mathcal{E}|$-vector and $R_s(t)$ be the K-vector associated with the residual arrival and service time information. The Markovian state descriptor will then be

$$\mathbf{Y}(t) = [Q(t);\ b(t);\ J(t);\ p(t);\ u(t);\ R_a(t);\ R_s(t);\ |Q(0)|], \tag{4.11}$$

and $\mathbf{Y}$ will represent the underlying state space. Imitating Dai's argument [6] and using the strong Markov property for piecewise deterministic processes of Davis [9], it is easy to show that the process $\{\mathbf{Y}(t),\ t \geq 0\}$ is a strong Markov process with state space $\mathbf{Y}$. A detailed description of such constructions can be found in Dai [6].

In the next two sections, we establish capacity realization under $\mathbf{DR}(r, l, \beta)$ in both open and closed network models.

5 Stability in open queueing networks

The main result regarding discrete-review policies in batch processing networks is that every such policy is stable for every open multiclass network satisfying **(A4)**. The theme of the proof is quite simple. By enforcing safety stocks and rigid processing plans, one ensures that the batch servers always run full batches. But these processing plans might result in unbounded WIP due to either too much waiting for the safety stocks to accumulate or too little stuff being pushed out in a review period. The idea is to then show that the average WIP remains bounded using the powerful technique of fluid limits.

Note that we do not distinguish between different reward functions at this point. The capacity realization results hold for *all* reward functions satisfying (4.3). The following theorem summarizes this result for open networks.

Theorem 5.1 *Let functions l, r and a vector β be as defined in section 4, satisfying (4.1)-(4.4). Then an open multiclass network is stable, as defined in Section 3.3, under the discrete-review policy* $\mathbf{DR}(r, l, \beta)$.

The proof of Theorem 5.1 easily follows from the analysis of discrete-review policies presented in [23, 22]. The primary technique used relies on the fluid limit approach pioneered by Dai [6]. There one examines the behavior of the *fluid limit model* under the scheduling policy under investigation, which is a deterministic and continuous approximation to the underlying network dynamics. This model is formally derived by a functional law-of-large-numbers type of scaling along any sequence of initial conditions $\{y^n\} \subset \mathbf{Y}$ such that $\|y^n\| \to \infty$ as $n \to \infty$. "Fluid" scaled processes are obtained by rescaling space and time by the initial population size as follows:

$$\bar{f}^n(t) = \frac{1}{\|y^n\|} f^{y^n}(\|y^n\| t). \tag{5.1}$$

The conditions satisfied by the limits of these processes determine the associated fluid limit model. Roughly speaking, Dai [6] established that proving stability of the limiting fluid model is sufficient to establish positive Harris recurrence of the underlying Markov chain. The interested reader is referred to [6] for further details.

The framework that has been described so far was developed for networks without batch processing resources. For our setup, given that the servers can process jobs in batches, one needs to formally extend this framework to this more general class of networks. It is a straight forward extension of Dai's treatment in [6, Theorem 4.1], to prove that for any sequence of initial condition $\{y^n\}$ such that $\|y^n\| \to \infty$, there exists a converging subsequence along which the fluid limits of the queue length and allocation processes exist. This follows from the fact that the allocation process is still Lipschitz continuous and that the jump size at every state transition is bounded above by $\max_i c_i$. Despite this result, the actual derivation of the fluid model equations in the presence of batch servers is problematic. The reason is that one needs to keep track of the batch sizes that get served at every point in time[4].

We can show, for the proposed class of discrete-review policies, that the servers process jobs in full batches with probability one, and thus the batch server can be replaced in fluid scale by a faster station serving one job at a time. Safety stock requirements are imposed so that within each short planning period each server can be fully occupied processing full batches of jobs. A large deviations analysis of the proposed discrete-review structure that establishes this claim can be found in [23, Lemmas 5.1-5.3] for the case of non-batch servers and can be extended to the case considered in this paper in a straight forward manner. As a parenthetical remark, it should be noted that even when the network is operating under heavy loading conditions, as for example in the network in Figure 1 for large values of N, the system is not guaranteed to operate its resources in full batches under an arbitrary policy.

The fluid model is then derived by first bounding the difference between the nominal and actual allocation processes and subsequently, obtaining the fluid limit of the nominal allocation process.

[4] In the absence of such information the fluid model equations can only be specified in the form of a differential inclusion, which provides a "loose" characterization in most cases.

The details of the proof were provided for the non-batch case in [23, Propositions 6.1-6.3] and are easily extended for our setup. Finally, stability is proved by specifying an appropriate Lyapunov function for the fluid model with the required negative drift. In specific, the functions used where $\phi(q(t))$ for the case of a constant reward vector ($r(q) \equiv r$) and $\Phi(q(t))$ for the general case of a dynamic reward rate function

$$\phi(q(t)) = r^T R^{-1} q(t) \quad \text{and} \quad \Phi(q(t)) = -\int_t^\infty e^{-\zeta(\tau - t)} r^T(q(\tau)) R^{-1} \dot{q}(\tau) d\tau, \tag{5.2}$$

where $\zeta > 0$ is a discount factor. Combining all of the results presented in this section and invoking Dai's stability Theorem, appropriately modified for batch processing networks, the proof of Theorem 5.1 is completed.

6 Efficiency in closed queueing networks

The treatment of discrete-review policies in closed queueing networks is new, and we note in passing that the results of this section have not been established even for single lot servers before. The basic idea of the proof remains the same as in the open network case. By enforcing safety stocks and rigid processing plans, the discrete-review policies ensure that the batch servers always run full batches. But this may result in very large throughput times because too few jobs are departing during a planning period, implying by Little's law that the throughput rate would be small for a given fixed WIP. The idea then is to establish that this does not happen, i.e., the rate of departures is sufficiently high using the fluid model approach.

First, we describe some necessary changes in the definition of the class of discrete-review policies in closed networks. In closed networks there are no external arrivals, that is $\lambda = 0$, and therefore one would need that first, $|\theta| \leq N$, where N is the total job population in the network and second, the following condition should be substituted for (4.7)

$$q - Rx \geq \theta. \tag{6.1}$$

The remaining of the planning logic is unaltered.

Similarly, in the infeasible planning logic the only modification required is the substitution of the following constraint in (4.10)

$$-Rx > \Gamma\beta + e. \tag{6.2}$$

Note that since P is assumed to be irreducible the infeasible planning linear program remains feasible under this constraint. The goal in closed networks is to redistribute the various jobs in the network so that the safety stock requirements are satisfied without being able to accumulate external arrivals. The rest of the infeasible planning logic remains unchanged.

Finally, the Markovian state descriptor needs to changed for the case of closed networks, namely by dropping R_a and noting that $\mathbf{Y}$ is restricted to the hyperplane $|Q(t)| = N$, where N is the fixed population size.

For closed network models, the equivalent notion of efficiency (as opposed to stability in the open network case) needs to be defined. Let $D(t)$ be the K-dimensional cumulative departure

process and let $\bar{D}(t) = M^{-1}\bar{T}(t)$ be the associated cumulative departure process in the fluid limit model. We define efficiency for the limit fluid model as follows.

Definition 6.1 *The fluid model associated with a closed multiclass network under a specified scheduling policy is* efficient *if for every class k, for every solution to the fluid model,*

$$\limsup_{t\to\infty} \frac{D_k(t)}{t} \geq \pi_k a^*. \tag{6.3}$$

Now we provide a sufficient condition for efficiency based on analysis of the associated fluid limits. Obtained by [19, Theorem 3.4.2], this is the analog of Dai's result for closed networks: the original closed network is efficient as defined in section 3.4 if the fluid model is efficient as defined above.

The following result is obtained for closed networks.

Theorem 6.1 *Let functions l, r and a vector β be as defined in section 4, satisfying (4.1)-(4.4). Then a closed multiclass network operating under the policy* $\mathbf{DR}(r, l, \beta)$ *is efficient as defined in (3.3).*

The first part of the proof of Theorem 6.1 uses the same kind of analysis as the one described in the previous section for the case of open network models. The first step is to establish that the fluid limit model under the policy $\mathbf{DR}(r, l, \beta)$ is given by the following set of equations:

$$\begin{aligned} \dot{q}(t) &= -Rv(t), \quad q(0) = z, & (6.4)\\ v(t) &\in \operatorname*{argmax}_{v\in\mathcal{V}(q(t))} r(q(t))^T v, & (6.5) \end{aligned}$$

where $\mathcal{V}(q(t)) = \{v : v \geq 0,\ Cv \leq e,\ (Rv)_k \leq 0 \ \text{ for all } k \text{ such that } q_k(t) = 0\}$. The proof of this result follows again from the analysis in [23, Propositions 6.1-6.3]. The next step in the proof is to analyze the resulting fluid model. This treatment is new, and hence is presented in some detail.

Let $f(t)$ be the instantaneous departure rate process in the fluid model, where f is mnemonic for flow. Then, $\bar{D}(t) = \int_0^t f(s)ds$. Equations (6.4)-(6.5) can be rewritten in terms of $f(\cdot)$ as follows:

$$\begin{aligned} \dot{q}(t) &= -(I - P^T)f(t), \quad q(0) = z, & (6.6)\\ f(t) &\in \operatorname*{argmax}_{f\in\mathcal{F}(q(t))} \tilde{r}(q(t))^T f, & (6.7) \end{aligned}$$

where $\mathcal{F}(q(t)) = \{f : f \geq 0,\ CMf \leq e,\ ((I - P^T)f)_k \leq 0 \ \text{ for all } k \text{ such that } q_k(t) = 0\}$ and $\tilde{r}(\cdot) = Mr(\cdot)$.

Let π be the unique invariant measure associated with the routing matrix P and recall the definition a^*. We claim that the departure rate vector $f = \lambda^*$, where $\lambda^* = a^*\pi$, is feasible for the linear program described in (6.7). Note that $f \geq 0$, by the definition of a^* it follows that $CMfx \leq e$, and finally since π is the invariant measure associated with P it follows that $(I - P^T)\pi = 0$. Hence,

$$\tilde{r}(q(t))^T f(t) \geq a^* \tilde{r}(q(t))^T \pi. \tag{6.8}$$

As noted earlier, in order to prove efficiency of the policy under investigation it is sufficient to prove that

$$\limsup_{t\to\infty} \frac{\bar{D}(t)}{t} \geq \lambda^* \quad a.s.. \tag{6.9}$$

Note that for all $t \geq 0$ we have that $\sum_k q_k(t) = 1$ which implies from (6.6) that

$$-e \leq P^T \bar{D}(t) - \bar{D}(t) \leq e. \tag{6.10}$$

Consider any subsequence of times $\{t_l\}$ such that $t_l \to \infty$ as $l \to \infty$ where the limit $\lim_{l\to\infty} \bar{D}(t_l)/t_l$ exists. ¿From (6.10) we have that

$$\lim_{l\to\infty} \left(P^T \frac{\bar{D}(t_l)}{t_l} - \frac{\bar{D}(t_l)}{t_l} \right) = 0. \tag{6.11}$$

Since π is unique, equation (6.11) implies that $\lim_{l\to\infty} \bar{D}(t_l)/t_l$ is of the form $a\pi$ for some positive constant a. ¿From (6.8) it follows that

$$\liminf_{l\to\infty} \frac{1}{t_l} \int_0^{t_l} \tilde{r}(q(s))^T (f(s) - a^*\pi) ds \geq 0,$$

which using the positivity (non-idleness) condition of $r(\cdot)$ in turn implies that $\pi(a - a^*) \geq 0$, or equivalently that $a \geq a^*$. Since, this is true for any converging subsequence $\{t_l\}$, condition (6.9) follows. To relate property (6.9) to efficiency for the closed network under the specified policy, we use Theorem 3.4.2 from [19]. This concludes the proof.

7 Choosing the reward vector and implementation

So far we have illustrated that batch servers can introduce unexpected subtle behavior that can restrain the system from realizing capacity, or equivalently, from sustaining its theoretical throughput capacity. Subsequently, a family of policies was described that is guaranteed to realize capacity both for open and closed network models under fairly general assumptions. Such policies are defined using a reward rate function $r(\cdot)$ that so far has not been specified, with the exception of some mild conditions required in the derivations of the analytical results presented. Naturally, the next step, which is left to the discretion of the system manager, is to specify the function $r(\cdot)$ according to the appropriate performance considerations, as well as other design and operational specifications. For example, for the case of semiconductor wafer fabs, which provided the initial motivation for this work and have been mentioned earlier, an excellent account of some of the relevant issues can be found in the review article by Uzsoy et al. [31], as well as the articles by Wein [32] and Kumar [16].

In the context of discrete-review policies, the choice of $r(\cdot)$ will affect the performance of the discrete-review policy as measured by the expected WIP at a given throughput satisfying **(A3)** in the open case, or by the throughput obtained at a fixed finite population in the closed case. The following is a list of some pointers as to the choice of the reward rate function that defines a discrete-review policy.

Non-idling: It is desirable that the discrete-review policy behave in a non-idling fashion. That is, it never idles a server which has enough work to do. Indeed, the condition $r(q) > 0$ for all $q \geq 0$ ensures that the associated fluid limits will be non-idling.

Mimicking priorities: From a system manager's point of view it would be useful if one could mimic the behavior of a specified (or desired) priority rule such as Shortest Remaining Process

Time (SRPT). The *relative magnitudes* of the reward rates assigned to the various job classes at every review point define a dynamic priority rule (or index rule) according to which we allocate resource usage. For example, the static reward vector $r_i \equiv i$ mimics the Last Buffer First Serve priority rule (which is the same as SRPT) in reentrant lines. The reader is referred to [23] for a detailed discussion of these first two points.

Fluid-scale asymptotic optimality: With discrete-review policies, it is possible to implement policies which are *optimal* in the following asymptotic sense. Consider a network optimization problem where we seek to minimize

$$J_\pi(z) = \mathbf{E}_z^\pi \int_0^T g(Q_k(t))dt, \tag{7.1}$$

where $g(\cdot)$ is a cost rate function, that could be for example a linear holding cost, and $\mathbf{E}_z^\pi$ denotes the expectation operator with respect to the probability measure $\mathbf{P}_z^\pi$ defined by any admissible policy π and initial condition z. The corresponding fluid optimization problem is defined by the following equation

$$\bar{V}_g(z) = \min_{v(\cdot)} \left\{ \int_0^T g(q(t))dt : q(0) = z \text{ and } (q,v) \in \text{fluid model} \right\}. \tag{7.2}$$

The optimal instantaneous allocation is a state feedback law that can be characterized by a direct application of dynamic programming principles (see for example Bertsekas [2]) as follows

$$v(t) \;=\; \operatorname*{argmin}_{v \in \mathcal{V}(q(t))} \nabla \bar{V}_g(q(t))^T(\lambda - Rv) \;=\; \operatorname*{argmax}_{v \in \mathcal{V}(q(t))} \nabla \bar{V}_g(q(t))^T Rv, \tag{7.3}$$

where $\mathcal{V}(q(t)) = \{v : v \geq 0,\ Cv \leq e,\ (Rv)_k \leq \lambda_k \text{ for all } k \text{ such that } q_k(t) = 0\}$ is the set of admissible controls when the state is $q(t)$. Define the vector valued function $r_g(q(t)) = R^T \nabla \bar{V}_g(q(t))$. This is a dynamic index rule or reward function, that defines the optimal policy for the fluid model. Using this reward rate function in a discrete-review policy will result in an asymptotically optimal policy under fluid scaling. That is, the associated fluid limits will be optimal under the criterion defined in (7.2). Policies designed using this asymptotic criterion may prove to be useful heuristic policies for a manufacturing. The reader is referred to [25] for motivation and definition of the property of fluid-scale asymptotic optimality and to [22] for a detailed discussion in the context of discrete-review policies.

From an implementation point of view, discrete-review policies have significant appeal. For example, one can think of the review period length as one shift (8 hours). Then, the system manager determines an aggregate processing plan for the entire shift by solving just one LP. Given the safety stock requirements imposed in the system, the execution of this plan is straight forward and more importantly, the low level details within each planning period are relatively insignificant. Thus, the operators can implement the plan without supervision throughout the shift. At the end of the shift, the new status of the manufacturing system is reviewed by the system manager and the process repeated. Finally, the detailed description of these control policies depends on just a few parameters that can be easily chosen and readjusted by the system manager through time.

Policy	Throughput
LBFS with no minimum batch size	0.357
LBFS with minimum batch size=10	0.5
Discrete-Review	0.5

Table 2: Comparison of policies in the deterministic case

8 A simulation experiment

Although we have provided mathematical proof of capacity realization under discrete-review polices, the performance of these policies as measured by the average WIP in the system at a given input rate $\lambda < \lambda^*$ in the open network case, or the throughput rate sustained at fixed WIP level in the closed network case remains unresolved. As a step towards resolving this issue, we revisit the example of Figure 1, and perform simulation studies of a discrete review policy in this example.

First consider the same setting as in Section 2, i.e., deterministic service times. For a closed network of fixed population $N = 1200$, we compare the LBFS policy with no minimum batch size, LBFS with a minimum batch size of 10, i.e., one cannot load less than ten customers to be processed by server 1 at any time, and $\mathbf{DR}((1,2,3)', 400, (1.25, 0.5, 1.25)')$, i.e., a discrete-review policy with a static reward function with $r_i \equiv i$, with review period of length 400, and $\beta = (1.25, 0.5, 1.25)'$ which corresponds to thresholds $\theta = (500, 200, 500)'$. Table 2 summarizes the results below. Note that the first row is exactly the same as the example considered in Section 2, where we have specialized to the case when $N = 1200$.

We now consider the case when the service times are distributed according to 4–Erlang distributions, but with the same mean as the deterministic service times considered in Figure 1. We can consider the three policies when the fixed population $N = 1200$ as described above. The results of 10 simulation runs of length $100,000$ time units each are tabulated in Table 3 below.

Policy	Throughput	95% Confidence Interval
LBFS with no minimum batch size	0.3325	5.2E-4
LBFS with minimum batch size=10	0.4994	6.2E-4
Discrete-Review	0.4949	6.3E-4

Table 3: Comparison of policies with 4–Erlang service time distributions

The results above verify that both LBFS with minimum batch size equal to 10 and the proposed discrete-review policy achieve throughputs close to the bottleneck throughput of 0.5 for moderate fixed population sizes N. It is also apparent that the former converges faster to the desired limiting throughput rate. This motivates a discussion of the speed of convergence of the throughput under the discrete-review policy to the bottleneck throughput as the fixed population increases.

8.1 Speed of convergence under discrete-review policies

One can calculate the speed of convergence for the throughput rate under the policy $\mathbf{DR}(r, l, \beta)$ by a simple calculation. Let $l_r(s)$ be the execution time of th r^{th} processing plan at server s. Let l be the expected duration of its processing plan. Then the execution time of its processing plan will be

$$l_r = \max(l_r(1), l_r(2)),$$

which in general will be greater than $E[l_r(1)] = E[l_r(2)] = l$. The quantity $l_r - l$ can be bounded using a large deviation argument and indeed, this is what was done in order to prove our theorems. Intuitively, from Chebyshev's inequality this difference is bounded by a small multiple of the standard deviation of the random variables $l_r(s)$. The latter is proportional to $\sqrt{l}$ and thus the throughput rate will be converging to its limit at a rate of $1/\sqrt{l}$ (since there are order l number of jobs in each processing plan). Note that the choice of the 4–Erlang distribution in the simulation study is equivalent to increasing l four-fold for exponential distributions as the both result in a four-fold reduction in the variance of the random variables $l_r(s)$.

This rate of convergence may appear to be slow and indeed one can verify that LBFS with minimum batch size equal to 10 will converge faster to bottleneck throughput than the proposed policy for the example in Figure 1. Nevertheless, this is not discouraging since the strength of the proposed family of policies is in that capacity realization is achieved for any multiclass queueing network, and for almost any dynamic reward rate function. In such a general setting simple heuristic rules cannot be guaranteed to be stable even for systems without batching, and as we mentioned earlier, minimum batch size rules will not realize capacity in general.

9 Concluding remarks

We have illustrated that the issue of capacity realization in systems with batch processing resources can be a difficult problem and we have described a general family of policies that overcomes this problem. The main feature of these policies is that they enforce class level safety stocks and rigid processing plans that effectively prevent unplanned idleness or inefficient use of capacity. Moreover, the only computation required in these policies is in determining the processing plans to be executed over each planning period, which is minimal.

This paper illustrates the power of discrete-review policies for solving problems of capacity realization in queueing networks. In fact, one could extend the network models that we consider, and still obtain similar results. For example, the models could be generalized in order to allow switchover times (or setup delays) when the servers switch between serving different job classes, routing control, as well as admission control. All these changes are virtually transparent in the planning and execution logic of a discrete-review policy, while the nature of the corresponding results will remain the same.

Ideally, the next step would be to simulate the policy against data collected form a large manufacturing system and compare performance with other existing scheduling policies.

Acknowledgments: We are grateful to Tomislav Galjanic for undertaking the simulation study

described in Section 8. We would also like to thank the anonymous reviewers for numerous excellent comments.

References

[1] Ahmadi, J. H., Ahmadi, R. H., Dasu, S., and Tang, C. S. Batching and scheduling jobs on batch and discrete processors. *Oper. Res. 39*, 4 (July 1992), 750–763.

[2] Bertsekas, D. P. *Dynamic Programming and Optimal Control, Vol.1.* Athena Scientific, Belmont, Massachusetts, 1995.

[3] Bertsimas, D., Gamarnik, D., and Tsitsiklis, J. N. Stability conditions for multiclass fluid queueing networks. *IEEE Trans. Aut. Control 41*, 11 (Nov. 1996), 1618–1631.

[4] Bramson, M. Instability of FIFO queueing networks. *Ann. Appl. Prob. 4*, 2 (1994), 414–431.

[5] Chen, H. Fluid approximations and stability of multiclass queueing networks: work-conserving policies. *Ann. Appl. Prob. 5* (1995), 637–655.

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

[7] Dai, J. G., and Meyn, S. Stability and convergence of moments for multiclass queueing networks via fluid limit models. *IEEE Trans. Aut. Control 40*, 11 (Nov. 1995), 1889–1904.

[8] Dai, J. G., and Weiss, G. Stability and instability of fluid models for certain re-entrant lines. *Math. Oper. Res. 21* (1996), 115–134.

[9] Davis, M. H. A. Piecewise-deterministic Markov processes: A general class of non-diffusion stochastic models. *J. R. Statist. Soc. Ser. B 46*, 3 (1984), 353–388.

[10] Deb, R. K., and Serfozo, R. F. Optimal control of batch service queues. *Ann. Appl. Prob. 5* (1973), 340–361.

[11] Glassey, C. R., and Weng, W. W. Dynamic batching heuristics for simultaneous processing. *IEEE Trans. Semiconductor Manufacturing 4* (1991), 77–82.

[12] Harrison, J. M. Brownian models of queueing networks with heterogeneous customer populations. In *Stochastic Differential Systems, Stochastic Control Theory and Applications*, W. Fleming and P. L. Lions, Eds., vol. 10 of *Proceedings of the IMA*. Springer-Verlag, New York, 1988, pp. 147–186.

[13] Harrison, J. M. The BIGSTEP approach to flow management in stochastic processing networks. In *Stochastic Networks: Theory and Applications*, F. Kelly, S. Zachary, and I. Ziedins, Eds. Oxford University Press, 1996, pp. 57–90.

[14] Harrison, J. M., and Nguyen, V. Some badly behaved closed queueing networks. In *Stochastic Networks*, F. Kelly and R. Williams, Eds., vol. 71. Proceedings of the IMA, 1995, pp. 21–29.

[15] Kumar, P. R. Re-entrant lines. *Queueing Systems 13* (1993), 87–110.

[16] Kumar, P. R. Scheduling semiconductor manufacturing plants. *IEEE Control Syst. Mag. 14*, 6 (Dec. 1994), 33–40.

[17] Kumar, P. R., and Meyn, S. P. Duality and linear programs for stability and performance analysis of queuing networks and scheduling policies. *IEEE Trans. Aut. Control 41*, 1 (Jan. 1996), 4–17.

[18] Kumar, S., and Kumar, P. R. Fluctuation smoothing policies are stable for stochastic re-entrant lines. *Discrete Event Dynamic Systems 6* (1996), 361–370.

[19] Kumar, S., and Kumar, P. R. Fluid limits and the efficiency of scheduling policies for stochastic closed reentrant lines in heavy traffic. In *Stochastic Networks: Stability and Rare Events*, K. Sigman et.al., Ed., vol. 117. Springer Lecture Notes in Statistics, 1996, pp. 41–64.

[20] Kumar, S., Schwerer, E., and Wood, S. C. Difficulties with capacity realization in semiconductor wafer fabs. In preparation.

[21] Lu, S. H., and Kumar, P. R. Distributed scheduling based on due dates and buffer priorities. *IEEE Trans. Aut. Control 36*, 12 (Dec. 1991), 1406–1416.

[22] Maglaras, C. Discrete-review policies for scheduling stochastic networks: Fluid-scale asymptotic optimality. *Ann. Appl. Prob.* (1997). Submitted.

[23] Maglaras, C. Dynamic scheduling in multiclass queueing networks: Stability under discrete-review policies. *Queueing Systems* (1997). Submitted.

[24] Medhi, J. Waiting time in a Poisson queue with a general bulk service rule. *Management Science 21* (1975), 777–782.

[25] Meyn, S. P. Stability and optimization of queueing networks and their fluid models. In *Mathematics of Stochastic Manufacturing Systems*, G. G. Yin and Q. Zhang, Eds., vol. 33 of *Lectures in Applied Mathematics*. American Mathematical Society, 1997, pp. 175–200.

[26] Miyazawa, M., and Taylor, P. G. A geometric product-form distribution for a queueing network with non standard batch arrivals and batch transfers. *Adv. Appl. Prob.* (1997). To appear.

[27] Neuts, M. F. A general class of bulk queues with Poisson input. *Ann. Math. Stat. 38* (1967), 369–373.

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

[29] Serfozo, R. F. Queueing networks with dependent nodes and concurrent movements. *Queueing Systems 13* (1993), 143–182.

[30] Spearman, M. L., Woodruff, D. L., and Hopp, W. J. CONWIP: a pull alternative to Kanban. *International Journal of Production Research 28*, 5 (1990), 879–894.

[31] Uzsoy, R., Lee, C. Y., and Martin-Vega, L. A. A review of production planning and scheduling models in the semiconductor industry part i. *IIE Trans. 24*, 4 (1992), 47–60.

[32] Wein, L. M. Scheduling semiconductor wafer fabrication. *IEEE Trans. Semiconductor Manufacturing 1*, 3 (Aug. 1988), 115–130.