Modeling and Analysis of Flexible Queueing Systems

Suryanarayanan Gurumurthi
Alcoa Technical Center,
Alcoa Center, PA 15069
suryan@me.umn.edu

Saifallah Benjaafar
Department of Mechanical Engineering
Division of Industrial Engineering
University of Minnesota
Minneapolis, MN 55455
saif@me.umn.edu

October, 2001

Abstract
We consider queueing systems with multiple classes of arrivals and heterogeneous servers where customers have the flexibility of being routed to more than one server and servers possess the capability of processing more than one customer class. We provide a unified framework for the modeling and analysis of these systems under arbitrary routing and server flexibility and for a rich set of control policies that includes customer/server-specific priority schemes for routing and queue selection. We use our models to generate several insights into the effect of system configuration. In particular, we examine the relationship between flexibility and throughput under varying assumptions for system parameters.
1. Introduction

In this paper, we consider the modeling and analysis of flexible queueing systems. We use the term flexible queueing systems to refer to systems with heterogeneous servers and multiple customer classes where customers have the flexibility of being routed to more than one server and servers possess the capability of processing more than one customer class. Customer classes can vary in demand rate and routing flexibility. Servers can vary in service rates and service flexibility. The routing of customers to servers is determined by a routing policy, that can be customer-specific, and the selection of the next customer to serve is determined by a queue selection policy, that can be server-specific. An example of a flexible queueing system is shown in Figure 1.

Flexible queueing systems arise in a variety of contexts, including manufacturing (Buzacott and Shanthikumar, 1992), telecommunication networks (Ross, 1995), computer systems (Kleinrock, 1976), and service operations (Hall, 1991). In manufacturing, there is often flexibility in routing jobs to functionally equivalent machines or production lines. These machines may vary by speed or cost. In telecommunication, flexibility arises from the availability of multiple links to which incoming calls can be routed. Different links may carry different capacities or provide different response times. Similar issues arise in large computer systems with multiple users and multiple servers. Flexibility derives from the ability to dynamically route users to different servers and to share computing capacity among different customers. Call centers are an important application in the service sector. Incoming customer calls vary by need, duration, and level of difficulty. Call centers are staffed by operators with varying skills who are capable of handling some or all of the call types (Koole and Mandelbaum, 2001).

In this paper, we provide a modeling framework for the analysis of general systems with an arbitrary number of servers and customer types and arbitrary routing and server flexibility. We consider a rich set of control policies that include strict priority schemes for customer routing and queue selection and dynamic policies such as longest queue first. Our models are applicable to systems with finite queue capacity. This queue capacity may be customer-specific or in the form of a global bound on work-in-process (WIP). An important contribution of this work is an alternative state representation scheme that not only is flexible but also yields significant economies in the size of the required state space. To our knowledge, this work is the first to
provide exact methods for the analysis of systems with general routing and server flexibility and heterogeneous servers.

To illustrate the usefulness of our models, we carry out a numerical study to examine the relationship between performance, as measured by throughput, and routing flexibility. Counter to intuition, we show that higher flexibility does not always improve throughput. For systems where it is beneficial, we show that the value of flexibility exhibits diminishing returns, with most of this value realized with relatively limited flexibility. In systems with symmetric demand rates and homogenous servers, we find that a special configuration called flexibility chaining yields most of the benefits of full flexibility. This is not the case in asymmetric systems, where we find that an asymmetric allocation of flexibility is generally superior. We examine the impact of control policies under different conditions of asymmetry. We show that there is a range of asymmetry in which the difference in throughput due to different control policies is maximum.

The remainder of this paper is organized as follows. In section 2, we provide a brief review of relevant literature. In Section 3, we present our model. In section 4, we comment on computational issues. In Section 5, we discuss numerical results and several insights. In section 6, we summarize our results and offer some concluding comments.

2. Literature Review

The literature on queueing systems with multiple servers can be broadly classified as pertaining to either design or control of these systems. For design, the literature can be grouped around four central questions: (1) how many servers should we have, (2) how should capacity be allocated to these servers, (3) how much flexibility should each server have, and (4) how much routing flexibility should we provide to each customer class. Issues pertaining to questions 1 and 2 are generally referred to as capacity allocation. The literature on this subject is voluminous. A review of important results and applications can be found in (Kleinrock, 1976) and (Buzacott and Shanthikumar, 1992). The available literature related to questions 3 and 4 focus mostly on comparing two extreme scenarios: dedication versus pooling. In a dedicated system, each customer class can be routed to only one server and each server can be routed to only one customer class. Hence, the system operates as a set of independent single server queues. In a pooled system, the customers are grouped in a single queue and can be routed to any server. In this case, the system operates as a multi-server queueing system. In systems with homogenous
service and demand distributions, it has been shown that a pooled system always outperforms a dedicated one (Smith and Whitt, 1981) (Benjaafar 1995). Calabrese (1992) shows that when average load is held constant in a $M/M/m$ queueing systems, average delay is strictly decreasing in $m$. Benjaafar (1995) offers performance bounds on the effectiveness of several pooling scenarios. Stecke and Solberg (1985) study pooling in the context of closed queueing network models of Flexible Manufacturing Systems (FMS) and show that throughput increases in the number of pooled servers.

In heterogeneous systems, pooling is not always superior. Buzacott (1996) considers a variety of pooling scenarios. He shows that when the service times of different customer classes are not identical in distribution, pooling can lead to longer queueing delays. He shows that the difference in performance between different pooling scenarios is sensitive to service time variability and the size of demand from each class. It is also affected by routing policy. These results support the observations made by Smith and Whitt (1981) who show that a pooled system can be made arbitrarily worse than a dedicated one through the introduction of a rare customer with long service times. Related discussion can be found in (Mandelbaum and Reiman, 1998).

Sheikhzadeh et al. (1998) propose server chaining as an alternative design strategy to both dedication and pooling. In a chained configuration, each customer can be routed to one of two neighboring servers and each server can process customers from two neighboring classes (see section 5). By maintaining an overlap of one customer between neighboring servers, Sheikhzadeh et al. show that chained systems, under the assumption of homogeneous demand and service times, achieve most of the benefits of total pooling. Similar insights were obtained by Graves and Jordan (1995) in a production planning context and by Hopp et al. (2001) for worker cross-training in serial production systems.

In this paper, we extend the analytical framework developed by Sheikhzadeh et al. (1998) to include a broader class of system configurations and control policies and a less restrictive set of assumptions. In particular, Sheikhzadeh et al. limit their analysis to three system configurations: dedication, chaining and full flexibility. They consider a symmetric system with equal number of customer classes and servers and identical bounds on queue sizes. They also consider control policies that do not allow for a full specification of a priority scheme for customer routing and queueing selection.
Control of flexible queueing systems can be broadly categorized as either static or dynamic. Under static control, customers are pre-assigned to specific servers according to a static partitioning scheme and served according to a fixed sequencing policy. Under dynamic control, routing and sequencing decisions are made based on the state of the system. Most of the literature on static control falls under the category of what is called workload allocation. In workload allocation demand that arises from different customer classes is partitioned among the different servers. The partitioning can be either discrete or continuous. In discrete partitioning, all of the demand from each customer must be allocated to one server. This gives rise to a combinatorial problem that is NP-hard in most cases. In continuous partitioning, fractional assignments are possible. The challenge in this case is to identify the optimal fractions of the demand from each customer class to assign to each server. A review of this literature can be found in Wang and Morris (1985).

The literature on dynamic control is vast (Sennot 1999). Much of this literature deals with systems consisting of a single class of arrivals and multiple servers with each server maintaining a separate queue. For systems with identical servers, Poisson arrivals and exponential processing times, Winston (1977) shows that the policy of routing an arriving customer to the shortest queue maximizes the discounted number of service completions in any finite interval \([0, T]\). Weber (1978) extends this result to systems with generally distributed inter-arrival times and servers with non-decreasing hazard rates. Hajeck (1984) treats a related problem with two servers and shows that the optimal policy is described by a switching function. A review of the dynamic routing literature can be found in Stidham and Weber (1993).

This paper deals with systems with finite buffers of which loss systems are a special case. Loss systems are queueing systems where waiting is not allowed. Loss systems with multiple servers and multiple customer classes have been widely studied in the context of telecommunication networks (Cooper, 1981). A recent review of relevant problems and literature can be found in Ross (1995).

3. A Modeling Framework for Flexible Queueing Systems

Let \(M = \{R_1, R_2, \ldots, R_m\}\) be the set of \(m\) servers in the system and \(P = \{p_1, p_2, \ldots, p_n\}\) be the set of \(n\) customer types. Possible customer-server assignments are denoted by an \(m \times n\) matrix \(A = [a_{ij}]\), where


\[
    a_{ij} = \begin{cases} 
        1, & \text{if part } p_i \text{ can be routed to resource } R_j; \\
        0, & \text{otherwise.} 
    \end{cases} \tag{3.1}
\]

We define a set of servers \( Q_i \) associated with each customer type \( p_i \), such that this customer type can be routed to any of the servers in \( Q_i \):

\[
    Q_i = \{ R_{i(1)}, R_{i(2)}, \ldots, R_{i(m_i)} \},
\]

where \( i(k) \) denotes the index of the \( k^{th} \) server assigned to customer type \( p_i \). We let \( \phi_i = |Q_i| = \sum_{j=1}^{m} a_{ij} \) denote the cardinality of set \( Q_i \). Similarly, we define \( T_j \) to be the set of customer types that can be routed to server \( R_j \) such that:

\[
    T_j = \{ p_{j(1)}, p_{j(2)}, \ldots, p_{j(n_j)} \},
\]

where \( j(k) \) denotes the index of the \( k^{th} \) customer type assigned to server \( R_j \) and \( \eta_j = |T_j| = \sum_{i=1}^{n} a_{ij} \) as the cardinality of set \( T_j \).

In addition to specifying the routing configuration, the analysis of flexible queueing systems requires the specification of routing and queue selection control policies. Routing policies involve selecting an idle server among all those capable of processing a customer upon the customer’s arrival to the system. Queue selection policies are needed to choose between different customer types in queue when a server capable of processing these customers becomes idle – i.e., selecting the next type to serve. WIP control policies determine how we assign limited queue capacity to the various customer types.

We consider stationary routing policies, where routing preferences can be specified in terms of a priority scheme for each customer type. For customer type and for each server, we associate a priority \( \alpha(p_i, R_j) \in \{ 1, 2, \ldots, m \} \), which, for notational compactness, we shall heretofore denote as \( \alpha_{ij} \). If there is competition between two or more idle servers for a customer of type \( p_i \), the customer is assigned to the server with the lower value of \( \alpha_{ij} \). Special cases of the priority scheme include the strict priority (SP) rule where \( \alpha_{ij} \neq \alpha_{ik} \) for all values of \( i, j \) and \( k \) \( (j \neq k) \) and the random routing (RR) rule where \( \alpha_{ij} = \alpha_{ik} \) for all values of \( i, j, \) and \( k \). In all cases, ties are broken arbitrarily.

For queue selection, we consider a dynamic policy under which a server always selects a customer from the longest queue from the set of feasible queues whenever it becomes available. Within each queue, customers are ordered on a first in first out (FIFO) basis. We term this policy
the longest queue first (LQF) policy. We also consider stationary queue selection policies, where queues are selected based on a priority scheme. Specifically, for each customer type and for each server, we associate a priority \( \gamma(p_i, R_j) \in \{1, 2, \ldots, n\} \), or more simply \( \gamma_{ij} \). If there is more than one feasible queue to choose from a server when \( R_j \) becomes available, a customer from the queue with the lowest value of \( \gamma_{ij} \) is selected. Special cases of priority schemes include the strict priority (SP) rule where \( \gamma_{ij} \neq \gamma_{ik} \) for all values of \( i, j \) and \( k \) \((j \neq k)\) and the random service (RS) rule where \( \gamma_{ij} = \gamma_{ik} \) for all values of \( i, j, \) and \( k \). Within each queue, customers are again ordered on a first in first out (FIFO) basis.

In this paper, we are concerned with the analysis of systems with finite queue capacity. We define queue capacity in one of two ways. A maximum queue size, denoted by \( w_i \), where \( w_i \geq 1 \) for \( i = 1, \ldots, n \), is associated with each customer type. Alternatively, we may specify a global bound on the maximum number of customers, regardless of type, that can be allowed in the system.

### 3.1 The State Space

The state of the system in a flexible queueing system can be described completely by specifying \((i)\) the number of customers in queue for each customer type, and \((ii)\) the state of every server in the set \( M \). Since we do not model server failures, there can only be two possible states, 0 and 1, for each server. The state of the system can thus be described using a vector \( N \equiv (n_1, n_2, \ldots, n_{n+m}) \), where \( n_i \) is the number of customers of type \( i \) for \( 1 \leq i \leq n \) and \( n_i \) is the state of server \( i \) for \( n+1 \leq i \leq n+m \). We denote the state space generated by such a representation as \( S_1 \). Although this state space representation could be used, it requires evaluating a large numbers of states even for small values of \( n \) and \( m \). In order to avoid such enumeration, Sheikhzadeh et al. (1998) took advantage of the simple structure of the systems they studied to redefine their state space. Since they considered only the three symmetric systems, specialization, chaining and full flexibility, where the number of customers equals the number of servers \((n = m)\), they were able to represent states of the system using a vector of state variables \( N = (n_1, n_2, \ldots, n_n) \) where, \( n_i = q_i + s_i \), with \( q_i \) \((q_i = 0, 1, \ldots, w_i)\) being the number of customers of the \( i^{th} \) type and \( s_i \) \((s_i = 0, 1)\) being the state of the \( i^{th} \) server. This modified state space representation yields considerable economies in the number of states that need to be considered since it reduces the number of state variables from \( n + m \) to \( n \).
In our case, because we allow the number of customers to be different from the number of servers and allow for asymmetries in the routing flexibility of customers and the customer flexibility of servers, the modified state space representation cannot be immediately used. However, in what follows we show that an alternative representation that yields significant reduction in the size of the state space is also possible here. The state representation we introduce shares some similarities with that of Sheikhzadeh et al. in that every state variable refers to either: (i) the number in queue for a customer type \( i \) \( (q_i) \), (ii) the state of a server \( j \) \( (s_j = 0, 1) \), or (iii) the sum of the two, \( q_i + s_j \), for a pair customer \( i \)-server \( j \).

Our approach is based on a graph-theoretic view of flexible queueing systems. In particular, we view our queueing system as a connected bipartite undirected graph, \( G(V, E) \), where \( V \) is the set of vertices and \( E \) is the set of edges. The vertices of the graph lie in two disjoint subsets. The first subset, \( V_1 \), corresponds to the set of customer types, and the second, \( V_2 \), corresponds to the set of servers. The edges of the graph connect the vertices in the set of customers to the vertices in the set of servers, where an edge \( e_{ij} \) between customer \( i \) and server \( j \) exists only if \( a_{ij} = 1 \). We limit our attention to connected graphs since a disconnected graph leads to two or more queueing systems that can be analyzed independently of each other. Before we present our state space representation scheme, we need the following two definitions and lemma.

**Definition 1:** A subgraph of \( G(V, E) \) in which every vertex has a degree of at most one (i.e., every vertex has at most one edge) is called a matching. The problem of finding such a subgraph is also sometimes called matching.

**Definition 2:** A maximal matching of graph \( G \ (V, \ E) \), is a matching \( G_x(V, \ E_x) \) such that \( |E_x| \geq |E_y| \) for any other matching \( G_y(V, \ E_y) \), where \( |E_x| \) and \( |E_y| \) refer to the cardinality of the set of edges \( E_x \) and \( E_y \) respectively.

**Lemma 1:** Consider an undirected and connected bipartite graph \( G(V, E) \) whose vertices can be partitioned into two disjoint sets \( V_1 \) with \( n \) vertices and \( V_2 \) with \( m \) vertices. Then, there exists a graph \( G_x(V, E_x) \) that has the following properties:

1. \( G_x(V, E_x) \) is a sub-graph of \( G(V, E) \),
2. \( G_x(V, E_x) \) is a maximal matching of \( G(V, E) \), and
3. \( G_x(V, E_x) \) has \( \min(n_1,n_2) \) edges and \( \max(m,n)-\min(m,n) \) unmatched vertices.

A proof of lemma 3.1 can be found in Cormen et al (1990). A maximal matching that has the above properties can be obtained using the Ford-Fulkerson max-flow algorithm (see for example
Cormen et al, 1990). If we let \( k = \min(m, n) \), then for our queueing system, a maximal matching would result in a subgraph where \( k \) customers and \( k \) servers are connected by an edge. Without loss of generality, we rename this set of customers and servers so that a customer that has been renamed \( p_i \) is connected to a server that has been renamed \( R_i \). Then, we can associate with this customer-server pair a state variable \( n_i \), where \( n_i = s_i + q_i \). The maximal matching also leads to \( l \) unmatched vertices (\( l = \max(m, n) - \min(m, n) \)). If \( m > n \), \( l = m - n \). The set of unmatched vertices consist in this case of \( l \) servers. These servers are then renamed \( R_{n+1}, R_{n+2}, ..., R_m \) and a state variable \( n_i = s_i, i=n+1, ..., m \), is associated with each. On the other hand, if \( n > m \), then the set of unmatched vertices corresponds to unmatched customers. We rename these customers \( p_{m+1}, p_{m+2}, ..., p_n \) and associate with each a state variable \( n_i \) where \( n_i = q_i, i=m+1, ..., n \). This process results in a state vector \( N = (n_1, n_2, ..., n_{n+m}) \), where

\[
 n_i = \begin{cases} 
 q_i + s_i & \text{if } 1 \leq i \leq k \\
 q_i & \text{if } k \leq i \leq n \text{ and } n > m \\
 s_i & \text{if } k \leq i \leq m \text{ and } m > n 
\end{cases}
\]

The process of generating the state vector \( N \) is illustrated in Figure 2 for an example system with 3 customers and 4 servers.

Our representation scheme which reduces the number of state variables from \((m + n)\) to \(\max(m, n)\) can have a significant impact on the number of states that need to be evaluated. For example, consider a system with total flexibility so that every customer type can be routed to any server and every server can process every customer. Then, it is not difficult to show that the number of states, using a representation with \( m + n \) state variables is given by

\[
|S_1| = 2^m - 1 + \prod_{i=1}^{n} (w_i + 1).
\]

In contrast, using our representation scheme the number of states is given by:

\[
|S_2| = 2^m - 1 + \prod_{i=1}^{n} w_i.
\]

Tables 1(a) and 1(b) illustrate the magnitude of the state space reduction achieved using our scheme. Note that the savings are increasing in the size of the problem (i.e., \( n, m, \) and \( w_i \)).
3.2 Performance Evaluation

In this section, we develop models for the performance evaluation of flexible queueing systems. Our approach begins by determining the probability of occurrence of each system state for the different control policies under consideration. From these probabilities, we show how to obtain various performance measures of interest. In doing so, we assume the following holds: (i) customers of type $i$ arrive according to an independent Poisson process with rate $\lambda_i$ and (ii) processing times at a server $j$ are exponentially distributed and iid with mean $1/\mu_j$. We model our system as a continuous time Markov chain (CTMC) with state vectors $N = (n_1, n_2, ..., n_q)$ where $q = \max(m, n)$ and $n_i$ is as defined in section 3.1. Our Markov chain is a birth-death process since the system transitions from its current state through either a single customer arrival or a single customer departure. The limiting probabilities of system states can be obtained from the balance equations of the Markov chain by equating the rate at which the system enters a state with the rate at which it leaves it (Ross, 1995). This relationship results in a set of linear equations that can be solved using a general-purpose linear equation solver.

Following the approach in Sheikhzadeh et al, we define for each state vector $N = (n_1, n_2, ..., n_q)$ two sets of states, type $a$ and type $d$, depending on whether an arrival or a departure causes the system to move to state $N$. We denote these sets of states as $N^a$ and $N^d$, respectively. Elements of $N^a$ ($N^d$) are state vectors $N_i^a$ ($N_i^d$), such that $n_i^a = n_i - 1$ ($n_i^d = n_i + 1$) and all other state variables having the same value as in $N$. We define $\delta(x)$ as a function that returns 1 if $x \geq 1$, and 0 otherwise. We also define

$$
v_i = \begin{cases} 
1, & \text{if } (1 \leq i \leq k) \text{ or } (k < i \leq n \text{ and } n > m) \\
0, & \text{otherwise},
\end{cases}
$$

and

$$
\omega_i = \begin{cases} 
1, & \text{if } (1 \leq i \leq k) \text{ or } (k < i \leq m \text{ and } m > n) \\
0, & \text{otherwise}.
\end{cases}
$$

The variable $v_i$ is used to indicate if a state variable $n_i$ includes the queue size of a customer $i$. Similarly, the variable $\omega_i$ is used to indicate if a state variable $n_i$ includes the state of a server $i$. Finally, we define the parameter $b_i$, the upper bound on the state variable $n_i$, as follows:
Whenever the system is in state $N$, it is straightforward to show that it leaves this state as a result of a service completion with rate $\sum_{i=1}^{q} \delta(n_i) \omega_i \mu_i$ and as a result of an arrival with rate $\sum_{i=1}^{q} \delta(b_i - n_i) \nu_i \lambda_i$. If we use $r^a_i$ ($r^d_i$) to denote the rates at which the system enters state $N$ from $N_i^a$ ($N_i^d$), and if we use notation $p(N)$, $p(N_i^a)$ and $p(N_i^d)$ to denote the steady state probabilities of those states, we can then write the Markov chain balance equation as follows:

$$\left( \sum_{i=1}^{q} \delta(n_i) \omega_i \mu_i + \sum_{i=1}^{q} \delta(b_i - n_i) \nu_i \lambda_i \right) p(N) = \sum_{N_i^a \in N^a} r^a_i p(N_i^a) + \sum_{N_i^d \in N^d} r^d_i p(N_i^d), \forall N \in S_2. \tag{3.6}$$

The set of linear equations in 3.6 along with the normalizing equation $\sum_{S_2} p(N) = 1$ form a set of $|S_2|$ simultaneous equations, which can be solved to determine the steady state probabilities, $p(N), N \in S_2$. However, in order to solve for $p(N)$, we need to first define the sets of entering states $N_i^a$ and $N_i^d$ and determine the associated rates $r^a_i$ and $r^d_i$, respectively.

### 3.2.1 The sets of entering states

There are two types of constraints that define whether a state can be included in either set $N^a$ or $N^d$. The first constraint deals with feasibility. Depending on the routing matrix and the control policies, there are certain states that can never occur. The second restriction stems from the requirement that it should be possible to go from a member of $N^a$ (or $N^d$) to $N$ by a one-step transition as a result of a single arrival or a single departure.

We first consider the set $N^a$. The system can move into state $N$ from $N_i^a$, only if $n_i^a = n_i - 1$ and all other state variables have exactly the same values as in $N$. That is,

$$n_i^a = n_i - 1 \text{ and } n_k^a = n_k \forall k \neq i. \tag{3.7}$$

The state $N_i^a$ exists only when one of the following mutually exclusive conditions holds:

$$1 \leq n_i^a \leq b_i - 1 \text{ and } n_i \neq 0, \forall l \neq i, i \in Q, \omega_i = \nu_i = 1; \tag{3.8}$$
\[ 0 \leq n_i^a \leq b_i - 1 \text{ and } n_i \neq 0, \forall l \neq i, l \in Q_i, \omega_i = 0, v_i = 1. \] (3.9)

\[ n_i^a = 0 \text{ and } n_r^a \leq 1, \forall r \neq i, r \in T_i, \omega_i = 1. \] (3.10)

Condition 3.8 states that for \( n_i^a \) (\( n_i = q_i + s_i \) since \( w_i = v_i = 1 \)) to increase by 1, there should already be customers of the same type in the queue and all the servers capable of processing customers of type \( i \) are busy. This follows from the fact that we do not allow a queue to form while a feasible server is idle. Note that \( n_i^a \) cannot be zero since server \( i \) must also be busy (\( s_i = 1 \)).

Condition 3.9 states that for \( n_i \) (\( n_i = q_i \) since \( w_i = 0 \)) to increase by 1, all the servers capable of processing customer \( i \) must be busy. In this case \( n_i \) can be zero since it represents customer queue size. Condition 3.10 considers cases when \( n_i \) (\( n_i = q_i + s_i \) or \( n_i = s_i \)) increases from 0 to 1. This is possible only if server \( i \) is idle which occurs only if there are no customers that can be processed on server \( i \) in the queue. From the above we can now define set \( N^a \) as follows:

\[ N^a = \{ N_1^a : 1 \leq n_i^a \leq b_i - 1 \text{ and } n_i \neq 0, \forall l \neq i, l \in Q_i, \omega_i = 1, v_i = 1; \]
\[ 0 \leq n_i^a \leq b_i - 1 \text{ and } n_i \neq 0, \forall l \neq i, l \in Q_i, \omega_i = 0, v_i = 1; \text{ or} \]
\[ n_i^a = 0 \text{ and } n_r^a \leq 1, \forall r \neq i, r \in T_i, \omega_i = 1; \text{ and } i = 1, \ldots, q \}. \]

Similarly, the system moves into a state \( N \) from a state \( N^d \) only if \( n_i^d = n_i + 1 \) and all other state variables maintain the same values as in \( N \). That is:

\[ n_i^d = n_i + 1 \text{ and } n_k^d = n_k, \forall k \neq i. \] (3.11)

It is possible to show that the above holds only if one of the following conditions is true:

\[ n_i^d = 1 \text{ and } n_r^d \leq 1, \forall r \neq i, r \in T_i, \omega_i = 1; \] (3.12)

\[ 2 \leq n_i^d \leq b_i \text{ and } n_i^d \neq 0, \forall l \neq i, l \in Q_i, \omega_i = 1, v_i = 1; \] (3.13)

\[ 1 \leq n_i^d \leq b_i \text{ and } n_i^d \neq 0, \forall l \neq i, l \in Q_i, \omega_i = 0. \] (3.14)

Condition 3.12 states that a departure would cause \( n_i^d \) to decrease from one to zero if there are no other customers available that can be processed on server \( i \). Condition 3.13 considers the case where \( n_i^d \geq 2 \) and follows from the fact that a queue of customer type \( i \) cannot form unless all the feasible servers are busy. Condition 3.14 is similar to 3.13 when \( n_i \) describes only the queue size of a customer type \( i \) (\( n_i = q_i \)). Set \( N^d \) is thus given by:

\[ N^d = \{ N_1^d : n_i^d = 1 \text{ and } n_r^d \leq 1, \forall r \neq i, r \in T_i, \omega_i = 1; \]

11
\[ 2 \leq n_i^d \leq b_i \text{ and } n_i^d \neq 0, \forall l \neq i, l \in Q_i, \omega_i = 1, v_i = 1; \]
\[ 1 \leq n_i^d \leq b_i \text{ and } n_i^d \neq 0, \forall l \neq i, l \in Q_i, \omega_i = 0, v_i = 1; i = 1, \ldots, q. \]

### 3.2.2 Transition rates \( r_i^a \) and \( r_i^d \)

In this section, we determine the transition rates for each of the routing and queue selection policy combinations we consider.

#### The SP-LQF System

When either condition 3.8 or 3.9 holds, a transition from \( N_i^a \) to \( N \) clearly occurs with rate:

\[ r_i^a = \lambda_i. \quad (3.15) \]

However, when condition 3.10 holds, the transition rate depends on the routing priorities. First note that for \( n_i^a \) to increase from zero to one, we need an arrival from a customer type that belongs to the set \( T_i \). Although necessary, this condition is not sufficient since an arrival of a customer of type \( p_k \) from the set \( T_i \) may not be routed to server \( i \) unless server \( i \) has the highest priority among those available to process customer \( p_k \). This means that the following condition

\[ \alpha_{ki} < \alpha_{kj} \text{ for all } j \text{ that satisfy } n_j = 0 \text{ and } w_j = 1. \]

must be satisfied. Since the arrival rate of customers of type \( k \) is \( \lambda_k \), the transition rate \( r_i^a \) is given by:

\[ r_i^a = \sum_{k \in T_i} \lambda_k \gamma(\alpha_{ki}), \]

where

\[ \gamma(\alpha_{ki}) = \begin{cases} 
1, & \text{if } \alpha_{ki}(1 - \delta(n_i)) \leq \alpha_{ki}, \forall k \in T_i, \forall t \in Q_t; \\
0, & \text{otherwise.} 
\end{cases} \quad (3.16) \]

Putting it all together, we have:

\[ r_i^a = \begin{cases} 
\lambda_i, & \text{if } 0 \leq n_i^a \leq b_i - 1 \text{ and } n_i^a \neq 0, \forall l \neq i, l \in Q_i, v_i = 1; \\
\sum_{k \in T_i} \lambda_k \gamma(\alpha_{ki}), & \text{if } n_i^a = 0 \text{ and } n_i^a \leq 1, \forall r \neq i, r \in T_i, \omega_i = 1. 
\end{cases} \]

Similarly, we can derive the transition rates \( N_i^d \) to \( N \). If condition 3.12 holds, then we clearly have \( r_i^d = \mu_i \). When condition 3.13 or 3.14 holds, the transition rate depends on the relative size
of the queues. There can be a transition from $N_i^{l}$ to $N$ if the queue for customer $p_i$ is one of the longest queues for any of the servers in the set $Q_i$. In other words, for a server $k$ in the set $Q_i$ to select queue $i$, queue $i$ must be one of the longest queues in the set $T_k$ (the set of feasible customers for server $k$). We denote by $B_k$ the number of customers that have the longest queue in the set $T_k$. When $B_k > 1$, queue $i$ is selected by server $k$ with probability $1/B_k$. Thus, the transition rate can be written as:

$$r_i^d = \begin{cases} 
\mu_i & \text{if } n_i^{l} = 1 \text{ and } n_q^{l} \leq 1, \forall q \neq i, q \in T_i, \omega_i = 1; \\
\sum_{k \in Q_i} \frac{\mu_k \epsilon_k(p_i)}{B_k} & \text{if } 2 \leq n_i^{l} \leq b_i \text{ and } n_l^{l} \neq 0, \forall l \neq i, l \in Q_i, \omega_i = 1; \\
\sum_{k \in Q_i} \frac{\mu_k \epsilon_k(p_i)}{B_k} & \text{if } 1 \leq n_i^{l} \leq b_i \text{ and } n_l^{l} \neq 0, \forall l \neq i, l \in Q_i, \omega_i = 1, \nu_i = 0.
\end{cases}$$

(3.17)

where $B_k$ is the number of customer types that have the longest queue in the set $T_k$, and $\epsilon_k(p_i) = 1$ if the queue of customer type $i$ is one of the $B_k$ longest queues in the set $T_k$, and 0 otherwise.

**The RR-LQF System**

The sets of entering states are the same for this queue selection policy as in the SP-LQF control policy. Only the transition rates differ. If an arrival of customer type $p_k$ occurs, where $p_k$ is of one of the customer types in set $T_i$, this arrival has an equal chance of being routed to any of the idle servers in the set $Q_i$. Thus,

$$r_i^a = \begin{cases} 
\lambda_i & \text{if } 1 \leq n_i^{a} \leq b_i - 1 \text{ and } n_l^{a} \neq 0, \forall l \neq i, l \in Q_i, \omega_i = 1, \nu_i = 1; \\
0 & \text{if } 0 \leq n_i^{a} \leq b_i - 1 \text{ and } n_l^{a} \neq 0, \forall l \neq i, l \in Q_i, \omega_i = 0, \nu_i = 1; \\
\sum_{k \in T_i} \frac{\nu_k \lambda_k}{\sum_{l \in Q_k} (1 - \delta(n_l^{a}))} & \text{if } n_i^{a} = 0 \text{ and } n_r^{a} \leq 1, \forall r \neq i, r \in T_i, \omega_i = 1.
\end{cases}$$

(3.18)

The rate $r_i^{d}$ is the same as in the SP-LQF case.

**The SP-RS System**

The transition rates $r_i^a$ are the same as in the SP-LQF policy. However, the rate $r_i^{d}$ is different. In particular, when condition 3.13 or 3.14 holds, a departure from any server $k$ in the
set $Q_i$ reduces the size of the queue of customer type $i$ with probability $1/\left\{ \sum_{r \in T_k} 1 - \delta(n_r) \right\}$. Thus, we have:

$$r_i^d = \begin{cases} 
\mu_i & \text{if } n_i^d = 1 \text{ and } n_r^d \leq 1, \forall r \neq i, r \in T_i, \omega_i = 1; \\
\sum_{k \in Q_i} \frac{\mu_k}{\sum_{i \in T_k} 1 - \delta(n_r)} & \text{if } 2 \leq n_i^d \leq b_i \text{ and } n_i^d \neq 0, \forall i \neq l, l \in Q_i, \omega_i = 1; \\
\sum_{k \in Q_i} \frac{\mu_k \varepsilon_k(p_i)}{B_k} & \text{if } 1 \leq n_i^d \leq b_i \text{ and } n_i^d \neq 0, \forall i \neq l, l \in Q_i, \omega_i = 0.
\end{cases}$$

(3.19)

The RR-RS System

The rate $r_i^d$ for the RR-RS is the same as in the RR-LQF policy, while the rate $r_i^d$ is the same as in the SP-RS policy.

The RR-SP System

The rate $r_i^d$ for the RR-SP policy is the same as in the RR-LQF policy, while the rate $r_i^d$ is given as follows. When conditions 3.12 holds, a transition from $N_i^d$ to $N$ clearly occurs with rate:

$$r_i^d = \mu_i.$$  

(3.20)

However, when either condition 3.13 or 3.14 holds, the transition rate depends on the strict priority scheme. First note that for $n_i^d$ to decrease by one, we need a departure from a server that belongs to the set $Q_i$. Although necessary, this condition is not sufficient since a departure from a server $R_k$ from the set $Q_i$ may not decrease $n_i^d$ by one, unless the customer type $i$ has the highest priority among all the customer types in queue that can be routed to $R_k$. This means that the following conditions must be satisfied:

$$\gamma_{ki} < \gamma_{kj} \text{ for all } j \text{ that satisfy } n_j \geq 2 \text{ and } v_j = 1, \omega_j = 1 \text{ or}$$

$$\gamma_{ki} < \gamma_{kj} \text{ for all } j \text{ that satisfy } n_j \geq 1 \text{ and } v_j = 1, \omega_j = 0,$$

Since the departure rate from server $R_k$ is $\mu_k$, the transition rate $r_i^d$ is given by:

$$r_i^d = \sum_{k \in Q_i} \mu_k \theta(\gamma_{ki})$$
where

\[
\theta(\gamma_{kl}) = \begin{cases} 
1, & \text{if } \gamma_{kl} (1 - \delta(n_l - 1)) \leq \gamma_{kl}, \forall k \in Q_i, \forall t \in T_k, \nu_l = \omega_l = 1; \\
\text{or } \gamma_{kl} (1 - \delta(n_l)) \leq \gamma_{kl}, \forall k \in Q_i, \forall t \in T_k, \omega_l = 0; \\
0, & \text{otherwise.}
\end{cases}
\] (3.21)

Putting it all together, we have:

\[
r^a_i = \mu_i, \begin{cases} 
\text{if } n^d_i = 1 \text{ and } n^d_r \leq 1, \forall r \neq i, r \in T_i, \omega_i = 1; \\
\sum_{k \in Q_i} \mu_k \theta(\gamma_{kl}), \text{ if } 2 \leq n^d_i \leq b_i \text{ and } n^d_i \neq 0, \forall l \neq i, l \in Q_i, \omega_i = 1, \nu_i = 1; \\
\text{or } 1 \leq n^d_i \leq b_i \text{ and } n^d_i \neq 0, \forall l \neq i, l \in Q_i, \omega_i = 0, \nu_i = 1 \\
0, & \text{otherwise.}
\end{cases}
\] (3.22)

The SP-SP System

The rate \( r^a_i \) for the SP-SP is the same as in the SP-LQF policy, while the rate \( r^d_i \) is the same as in the RR-SP policy.

3.3 Systems with a Global Bound on WIP

Our analysis can be extended to system where there is a global, instead of customer-specific, bound on work-in-process. This means that the maximum number of customers that can be allowed in the system (regardless of type) is \( b \). With this requirement, it is not difficult to show that a transition from a state \( N^a_i \) occurs only if one of the following conditions holds:

\[
n^a_i \geq 1, \sum_{i=1}^q n_i < b, \text{ and } n_i \neq 0, \forall l \neq i, l \in Q_i, \omega_i = 1, \nu_i = 1; \] (3.23)

or

\[
n^a_i \geq 0, \sum_{i=1}^q n_i < b, \text{ and } n_i \neq 0, \forall l \neq i, l \in Q_i, \omega_i = 0, \nu_i = 1. \] (3.24)

\[
n^a_i = 0 \text{ and } n^d_r \leq 1, \forall r \neq i, r \in T_i, \omega_i = 1. \] (3.25)

Similarly, a transition from \( N^d_i \) to \( N \) occurs only if one of the following conditions is true:

\[
n^d_i = 1 \text{ and } n^d_r \leq 1, \forall r \neq i, r \in T_i, \nu_i = 1. \] (3.26)

\[
2 \leq n^d_i \leq b_i \text{ and } n^d_i \neq 0, \forall l \neq i, l \in Q_i, \omega_i = 1; \] (3.27)
\[ 1 \leq n_i^d \leq b_i, \text{ and } n_i^d \neq 0, \forall l \neq i, l \in Q_i, \omega_i = 0. \] 

(3.28)

The transition rates \( r_i^a \) (\( r_i^d \)) associated with conditions 3.23-3.25 (3.26-3.28) are the same as those associated with conditions 3.8-3.10 (3.12-3.14) of the previous section for each pair of routing and queue selection policies we consider.

### 3.4 Performance Measures

From \( p(N) \), we can obtain the marginal probability \( p(n_i) \) associated with the state variable \( n_i \).

Our primary measure of performance is throughput for each customer type \( i \) which can be obtained as follows:

\[
\tau_i^p = \lambda_i (1 - p(n_i = b_i)) \tag{3.29}
\]

from which, we can then obtain total system throughput as:

\[
\tau_s = \sum_{i=1}^{n} \tau_i^p. \tag{3.30}
\]

We can also derive expressions for throughput due to each server \( j \) as:

\[
\tau_j^R = \mu_j (1 - p(n_j = 0)). \tag{3.31}
\]

Several other measures of performance can be obtained as well, including expected queue size for each customer type, average utilization of each server, expected total WIP in the system, and expected flow time.

### 3.5 Validation of the Modeling Framework

In order to validate the model, we carried out computer Monte-Carlo simulations of several example systems. Table 2 shows a representative set of simulation results for an example system with 5 servers and 5 customer types arranged in a chained configuration (see section 5 for a discussion of chaining). The value of system parameters are \( \mu_j = 1 \) for \( j = 1, \ldots, 5; \lambda_1 = 1, \lambda_2 = 1.4, \lambda_3 = 1.2, \lambda_4 = 1.1, \lambda_5 = 0.7; \) and \( b_i = 4 \) for \( i = 1, \ldots, 5 \). The data shown is for systems operating under the SP routing policy and the LQF queue selection policy. The results are from 15 replications of 100,000 arrivals to the system with different random number seeds used for every replication.
4. **Some Remarks on Computational Issues**

Computing the stationary distribution of a finite irreducible Markov Chain is equivalent to solving a system of linear equations. Although the problem is mathematically simple, the number of states that the system can occupy could make it computationally challenging. In our application, the number of states can be large even for systems with a relatively small number of servers and customer classes. There is a vast literature on computational methods for solving Markov chains – see for example Stewart (1994) for a comprehensive treatment and literature review. These methods can be broadly classified into two categories: *iterative* and *direct*. Iterative approaches are by far the most commonly used. In addition to being simple to implement, iterative methods offer significant economies in computer memory usage. Iterative methods can also take advantage of good initial estimates of the solution vector. More importantly, an iterative process can be halted once a desired amount of accuracy is obtained. Finally, with the iterative method, the buildup of rounding error is minimized since the infinitesimal generator matrix is never altered during calculations. However, iterative methods have a major drawback in that often they require a long time to converge to the desired solution. Examples of iterative methods include the power method, the Gauss-Siedel iteration method, and the successive over-relaxation method (Heyman, 1987) (Stewart, 1994).

An alternative to the iterative approach is the direct approach. A major advantage of the direct approach is enhanced solution accuracy. The simplest direct approach is Gaussian elimination. However, Gaussian elimination can be unstable for certain classes of problems. A more stable variant of Gaussian elimination is the widely used GTH algorithm, developed by Grassman, Taksar, and Heyman (1998). Although the GTH algorithm requires more numerical operations than standard implementations of Gaussian elimination, it can offer a significant improvement in accuracy when the infinitesimal generator matrix is ill-conditioned. Problems arise with the GTH algorithm when memory is limited. Computational times can also be significant for large problems. In environments with tight constraints on computational times, recent implementations of the GTH algorithm on parallel processors have been encouraging – see for example (Cohen et al, 1997).

In this paper, and for the purpose of generating numerical results in section 5, we used CPLEX, a commercial solver available from the ILOG corporation. Using CPLEX requires reformulating and solving our problem as a Linear Program (LP). For the cases we solve,
Cplex provides solutions with reasonable accuracy and speed. This is illustrated in Tables 3(a) and 3(b). In Table 3(a) computational and accuracy are shown for a fully flexible system with four servers and four customer types. Each of the scenarios corresponds to a different buffer size which ranges from 1 to 13. In Table 3(b), results are shown for systems where we vary the number of servers and customers from 1 to 12 for a fixed buffer size of 2. Computations were carried out on a Pentium-III microprocessor with a 650 MHz frequency and 256MB of memory.

5. Numerical Examples and Insights

In this section, we use numerical examples to generate some insights into the behavior of flexible queueing systems. We focus on the role flexibility plays and how it interacts with other system parameters. The treatment is not meant to be comprehensive but simply to illustrate how the model can be useful in developing better understanding of how flexible queueing systems behave. We make four main observations. First, we show that additional flexibility without a well-designed control policy can reduce throughput. Second, for symmetric systems, we show that a chained configuration of routing flexibilities provides most of the throughput benefits of total flexibility. We also show that increased chaining exhibits diminishing returns. Third, for symmetric systems, we show that throughput is maximum when flexibility is balanced among the various customers and the servers. However, we show that this is not necessarily the case for asymmetric systems with heterogeneous servers or asymmetric demand rates. For these systems, we find that chaining is not always superior to a non-chained configuration. Fourth, we examine the impact of control policies under different conditions of asymmetry. We show that there is a range of asymmetry in which the difference in throughput due to different control policies is maximum.

Observation 1: Increasing routing flexibility for one or more customers can reduce system throughput.

Consider the five scenarios shown in Figure 3. The scenarios correspond to systems with varying routing flexibility. Values of system parameters are as follows: \( \mu_j = 1 \) for \( j = 1, 2, ..., 5 \); \( \lambda_1 = 5.0, \lambda_2 = 1.25, \lambda_3 = 0.75, \lambda_4 = 0.5, \lambda_5 = 0.25 \); and \( b_i = 2 \) for \( i = 1, ..., 5 \). The queue selection policy is SP with the customer with the higher arrival rate assigned a lower priority. The server selection is also of the SP type with servers ordered per the priority scheme \( \alpha_1 < \alpha_2 < \alpha_3 < \alpha_4 < \alpha_5 \). System throughput corresponding to each scenario is shown in Table 4(a). In each scenario, we
increase the routing flexibility of one or more customers. As we can see, any increase in flexibility leads to a reduction in system throughput. The amount of this decrease is sensitive to whose routing flexibility, among the customers, is increased. These effects can be explained as follows. In each scenario, greater flexibility is achieved by allowing more customers to be routed to server $R_1$. This increases the overall loading of $R_1$ and increases the probability of balking for all items that were initially assigned to it (e.g., $p_1$). In turn, this leads to lower throughput for these customers, which may not always be matched by increased throughput for customers whose flexibility has been increased. This would be particularly the case when the demand rates for the original customers are high or when the flexibility of the newly assigned customer is already high. Clearly, using a better control policy could mitigate these effects. For example, as shown in Table 4(b), using the LQF instead of the SP policy in selecting customer queues tends to dampen the above effects. These effects would altogether disappear if a state-based control policy were used.

Several recent studies (Hopp et al., 2001), (Jordan and Graves, 1995), and (Sheikhzadeh et al., 1998) have shown in a variety of contexts that a chained configuration in which each customer can be routed to one of two neighboring servers and each server can process customers from two neighboring classes achieves most of the benefits of total flexibility. In the following observation, we confirm that this is true for symmetric systems (i.e., systems with identical servers, customer classes, and queue sizes). We also show that higher order chaining exhibits diminishing returns. A system with chaining of order $k$ ($k \geq 2$) refers to systems where each customer can be routed to $k$ neighboring servers and each server can process $k$ neighboring customers. Systems with varying orders of chaining are shown in Figure 4.

**Observation 2:** In a symmetric system, increased chaining exhibits diminishing returns with most of the value of total flexibility achieved with the initial chain.

To illustrate the above result, consider the system scenarios shown in Figure 4. The scenarios correspond to systems with progressively higher orders of chaining. Scenario 1 corresponds to a system with no flexibility and scenario 6 to a system with total flexibility (chaining of order $k=6$). For each scenario, we obtain the corresponding throughput. Representative results are shown in Figure 5 (system parameters for the data we show are: $\mu_j = 1$ for $j = 1, ..., 6$; $\lambda_i = 1$ and $b_i = 2$ for $i = 1, ..., 6$, and the control policy is SP-LQF). As we can see, most of the increase in throughput is achieved by forming the initial chain. Higher order
chaining increase throughput only marginally and in progressively diminishing amounts. These results suggest that total flexibility, which corresponds to chaining of maximal order, is generally unjustified if increases in flexibility require significant investments. Note that the effect of chaining, and more generally flexibility, is most significant when utilization is in the mid-range. When utilization is low, customers are rarely blocked regardless of flexibility. On the other hand, when utilization is high, throughput is close to its maximum value since servers are always busy.

Although increases in chaining exhibit diminishing returns, this is not true with increases in flexibility in general. In Figure 6, we illustrate the effect of gradually increasing routing flexibility one customer at a time (i.e., adding a single arc at a time to the graph). In this case, each level of chaining is attained via a series of increases to the number of arcs. From Figure 7, we see that, surprisingly, the marginal increase in throughput is not always decreasing in the number of arcs. In fact, within each level of chaining, it exhibits increasing returns, with the last link in each chain realizing the largest marginal gain in throughput. This result further underscores the importance of forming fully connected chains. This is different from the observation made in Hopp et al. (2001) for serial systems, where only the last link in the chain is found to exhibit increasing returns.

Observation 3: In a symmetric system, a balanced allocation of routing flexibility among the customers leads to higher throughput.

Consider the four system scenarios shown in Figure 8. The scenarios correspond to systems where the total amount of routing flexibility (i.e., the number of edges connecting customers to servers) remains fixed but the allocation of this flexibility (edges) is varied. System throughput for varying levels of system is shown in Figure 9 (values of system parameters, unless noted, are the same as in the previous observation; the results are shown for the SP-LQF policy with similar effects are observed for other policies). It is clear from the results that increased balancing in routing flexibility leads to higher throughput. Here again, the effect is most significant when system loading is in the mid-range.

A balanced allocation is however not always optimal. In fact, as we note in the following observation, an asymmetric allocation of flexibility can be more beneficial when there is asymmetry in customer demand rates.

Observation 4: In systems with asymmetric demand rates, an asymmetric allocation of flexibility can (but not always) lead to higher throughput.
We illustrate the above by comparing the performance of the two flexibility scenarios shown in Figure 10. In order to examine the effect of demand asymmetry, we vary the fraction of the demand due to customer class $p_1$ while keeping the total demand on the system constant. We assume the demand rate for customers 2, 3 and 4 are maintained equal. In this fashion, a simple measure of demand asymmetry is the ratio of the demand rate for customer 1 to the demand rate of any of the other customers. We denote this ratio by $V$ and vary from 1 to 5. The results are shown in Figure 11. We can clearly see that when demand is sufficiently asymmetric ($V > 2$), scenario 2 leads to higher throughput. This effect is more significant in the more asymmetric systems. Note also that while higher demand asymmetry decreases throughput in a balanced system, throughput actually increases with demand asymmetry in the unbalanced one.

An important implication of the above is that chaining is not always superior to non-chaining. In fact, in asymmetric systems, we find that there is always a non-chained configuration that performs better than chaining. It is tempting to argue that, in asymmetric systems, customers with the higher demand rates should be assigned greater flexibility. However, we find that this is not always true. The optimal allocation of flexibility generally depends on both customer demand rates as well as the capacity of the servers to which customers are routed. To see this, consider the extreme case where the customer with the highest demand rate can be routed to only one server but this server has nearly zero mean processing times. In this case, clearly, providing more flexibility to this customer will do little to improve throughput. In general, we find that the optimal allocation of flexibility involves a complex relationship between customer demand rates and server capacities.

Finally, we note that asymmetry affects the difference in performance between different control policies. In particular, there is a range of asymmetry, in which this difference is maximum.

**Observation 5:** The difference in performance between different queue selection policies is affected by demand asymmetry. There is a level of asymmetry under which this difference is maximum.

We consider a system with three servers and three customer types. We vary the asymmetry in the demand rates of the different customers by increasing the relative contribution of different customer types to total demand while maintaining total demand constant. We use
as our measure of demand asymmetry. We compare two queue selection policies, \( SP_1 \) and \( SP_2 \). Under policy \( SP_1 \), we give highest priority to the queue with the lowest demand rate (queue 3).

Under \( SP_1 \), we give highest priority to the queue with the highest demand rate (queue 1). The difference in throughput between the two policies is shown in Figure 12. As we can see, the difference is maximum when asymmetry is in the mid-range.

A similar effect is observed with respect to routing policies and asymmetries in processing rate. In that case, we find that the impact of routing policies is insignificant where processing rate asymmetry is either very high or very low. It is however, significant when the asymmetry is in the mid-range.

6. Concluding Comments

In this paper, we presented a framework for the representation, modeling and analysis of flexible queueing systems. The analytical model allows for the analysis of general system configurations with an arbitrary number of customers and servers, an arbitrary routing matrix, asymmetric demand and processing rates, asymmetric bounds on customer queue sizes, and a wide range of control policies. The models are generic and can be used to analyze flexible queueing systems in a variety of applications. They can also serve as a decision support tool for the planning and design of these systems. Furthermore, our characterization of the probability distribution of system states and the transition probability between these states offers the opportunity to formulate optimal control problems (e.g., using the framework of a Markov decision process).

Our model can be extended in a variety of ways. This includes relaxing the assumptions of Poisson demand and exponential processing times and allowing service times to vary by customer and server. It would then be useful to examine the impact of demand and service variability on different system configurations and different control policies. In many applications, such as manufacturing, the processing of multiple customers on the same servers is accompanied by losses in efficiencies due to switchover times or costs. The current model could be extended to account for these inefficiencies. In other applications, such as telecommunication networks, the processing of a customer requires the simultaneous contribution of more than one
server. For these applications, there is a need to extend the analysis to systems where customers consume varying amounts of capacity.

Acknowledgments: The authors are grateful to William Cooper and Mike Taaffe for many useful comments on an earlier draft. The research of the second author is supported by NSF through grants DMII-9908437 and 9988721.
References


Table 1(a) – Percentage reduction in the size of the state space
\((w_i = 1 \text{ for } i = 1, 2, \ldots, n)\)

| \(n\) | \(m\) | \(|S_1|\) | \(|S_2|\) | Percentage difference |
|---|---|---|---|---|
| 3 | 3 | 132 | 71 | 46.21 |
| 4 | 4 | 640 | 271 | 57.66 |
| 5 | 5 | 3156 | 1055 | 66.57 |
| 6 | 6 | 15688 | 4159 | 73.49% |
| 7 | 7 | 78252 | 16511 | 78.90 |
| 8 | 8 | 390880 | 65791 | 83.17 |
| 9 | 9 | 1953636 | 262655 | 86.56 |
| 10 | 10 | 9766648 | 1049599 | 89.25 |

Table 1(b) - Percentage reduction in the size of the state space
\((n=m=6)\)

<table>
<thead>
<tr>
<th>(w_i)</th>
<th>(S_1)</th>
<th>(S_2)</th>
<th>Percentage difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>792</td>
<td>127</td>
<td>83.96</td>
</tr>
<tr>
<td>3</td>
<td>4159</td>
<td>792</td>
<td>80.96</td>
</tr>
<tr>
<td>4</td>
<td>15688</td>
<td>4159</td>
<td>73.49</td>
</tr>
<tr>
<td>5</td>
<td>46719</td>
<td>15688</td>
<td>66.42</td>
</tr>
<tr>
<td>6</td>
<td>117712</td>
<td>46719</td>
<td>60.31</td>
</tr>
<tr>
<td>7</td>
<td>262207</td>
<td>117712</td>
<td>55.11</td>
</tr>
<tr>
<td>8</td>
<td>531504</td>
<td>262207</td>
<td>50.67</td>
</tr>
<tr>
<td>9</td>
<td>1000063</td>
<td>531504</td>
<td>46.85</td>
</tr>
<tr>
<td>10</td>
<td>1771624</td>
<td>1000063</td>
<td>43.55</td>
</tr>
</tbody>
</table>
### Table 2 - Simulation results

<table>
<thead>
<tr>
<th>Replication</th>
<th>$\tau_1^R$</th>
<th>$\tau_2^R$</th>
<th>$\tau_3^R$</th>
<th>$\tau_4^R$</th>
<th>$\tau_5^R$</th>
<th>Maximum Half Width</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.92103</td>
<td>0.96387</td>
<td>0.97281</td>
<td>0.95905</td>
<td>0.91318</td>
<td>0.00592</td>
</tr>
<tr>
<td>2</td>
<td>0.91737</td>
<td>0.9631</td>
<td>0.97174</td>
<td>0.95837</td>
<td>0.91314</td>
<td>0.00527</td>
</tr>
<tr>
<td>3</td>
<td>0.92196</td>
<td>0.96418</td>
<td>0.97335</td>
<td>0.96144</td>
<td>0.91712</td>
<td>0.00518</td>
</tr>
<tr>
<td>4</td>
<td>0.9213</td>
<td>0.9638</td>
<td>0.9736</td>
<td>0.96048</td>
<td>0.91543</td>
<td>0.00494</td>
</tr>
<tr>
<td>5</td>
<td>0.92324</td>
<td>0.96376</td>
<td>0.97246</td>
<td>0.96082</td>
<td>0.91764</td>
<td>0.00482</td>
</tr>
<tr>
<td>6</td>
<td>0.92483</td>
<td>0.96396</td>
<td>0.97366</td>
<td>0.95973</td>
<td>0.91756</td>
<td>0.00402</td>
</tr>
<tr>
<td>7</td>
<td>0.92177</td>
<td>0.96368</td>
<td>0.97276</td>
<td>0.96234</td>
<td>0.91883</td>
<td>0.00401</td>
</tr>
<tr>
<td>8</td>
<td>0.92007</td>
<td>0.96557</td>
<td>0.97527</td>
<td>0.96206</td>
<td>0.91607</td>
<td>0.00528</td>
</tr>
<tr>
<td>9</td>
<td>0.92073</td>
<td>0.96451</td>
<td>0.97376</td>
<td>0.96097</td>
<td>0.91765</td>
<td>0.0044</td>
</tr>
<tr>
<td>10</td>
<td>0.91668</td>
<td>0.96265</td>
<td>0.97322</td>
<td>0.95911</td>
<td>0.91361</td>
<td>0.00519</td>
</tr>
<tr>
<td>11</td>
<td>0.91909</td>
<td>0.96264</td>
<td>0.97087</td>
<td>0.95818</td>
<td>0.91285</td>
<td>0.00478</td>
</tr>
<tr>
<td>12</td>
<td>0.91918</td>
<td>0.96416</td>
<td>0.97244</td>
<td>0.96141</td>
<td>0.91633</td>
<td>0.00558</td>
</tr>
<tr>
<td>13</td>
<td>0.92213</td>
<td>0.96459</td>
<td>0.97558</td>
<td>0.96326</td>
<td>0.9169</td>
<td>0.00519</td>
</tr>
<tr>
<td>14</td>
<td>0.91899</td>
<td>0.96296</td>
<td>0.97175</td>
<td>0.95964</td>
<td>0.91322</td>
<td>0.00533</td>
</tr>
<tr>
<td>15</td>
<td>0.92163</td>
<td>0.96299</td>
<td>0.97368</td>
<td>0.95837</td>
<td>0.91288</td>
<td>0.00503</td>
</tr>
<tr>
<td>Mean</td>
<td>0.920667</td>
<td>0.963761</td>
<td>0.97313</td>
<td>0.960349</td>
<td>0.915494</td>
<td>0.004996</td>
</tr>
<tr>
<td>Standard deviation</td>
<td>0.002156</td>
<td>0.000805</td>
<td>0.001252</td>
<td>0.00158</td>
<td>0.002134</td>
<td>0.00053</td>
</tr>
<tr>
<td>Analytical results</td>
<td>0.915075</td>
<td>0.967036</td>
<td>0.970871</td>
<td>0.962943</td>
<td>0.914471</td>
<td>-</td>
</tr>
</tbody>
</table>
Table 3(a) - Solution Times and Accuracy for systems with varying buffer sizes

| $|S_2|$ | CPU Time (seconds) | $\sum P(N)$ |
|-----|-------------------|-------------|
| 16  | 0.01              | 1.00000004 |
| 31  | 0.04              | 0.99999923 |
| 96  | 0.07              | 0.999999596 |
| 271 | 0.09              | 0.999999606 |
| 640 | 1.01              | 0.999998642 |
| 1311| 1.57              | 0.99999999  |
| 2416| 4                 | 1.00000015  |
| 4111| 9.5               | 1.00000019  |
| 6576| 34.2              | 1.00000012  |
| 10015| 91.7             | 0.999999134 |
| 14656| 186.4            | 0.99999999  |
| 20751| 312.2            | 1.00000021  |
| 28576| 1025.1           | 1.00000017  |

Table 3(b) - Solution times and accuracy for systems with varying number of servers and customers

| $|S_2|$ | CPU Time (seconds) | $\sum P(N)$ |
|-----|-------------------|-------------|
| 15  | 0.01              | 1.000000017 |
| 31  | 0.04              | 1.000000014 |
| 63  | 0.06              | 1.00000045  |
| 127 | 0.07              | 0.99999929  |
| 255 | 1.09              | 1.000000056 |
| 511 | 0.94              | 1.00000069  |
| 1023| 1.21              | 0.999999512 |
| 2047| 3.52              | 0.99999999  |
| 4095| 9.32              | 1.000000011 |
| 8191| 86.9              | 1.000000024 |
| 16383| 262.3            | 0.99999931  |
| 32767| 1407.4           | 1.00000069  |
Table 4(a) - The effect of higher flexibility on throughput

<table>
<thead>
<tr>
<th>Scenario</th>
<th>$\tau_1^R$</th>
<th>$\tau_2^R$</th>
<th>$\tau_3^R$</th>
<th>$\tau_4^R$</th>
<th>$\tau_5^R$</th>
<th>$\tau_s$</th>
<th>$\tau_1^P$</th>
<th>$\tau_2^P$</th>
<th>$\tau_3^P$</th>
<th>$\tau_4^P$</th>
<th>$\tau_5^P$</th>
<th>$\tau_s$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scenario 1</td>
<td>0.967741</td>
<td>0.99253</td>
<td>0.654998</td>
<td>0.468458</td>
<td>0.228868</td>
<td>3.3126</td>
<td>0.967742</td>
<td>0.6471</td>
<td>0.698342</td>
<td>0.57678</td>
<td>0.422632</td>
<td>3.3126</td>
</tr>
<tr>
<td>Scenario 2</td>
<td>0.978557</td>
<td>0.610363</td>
<td>0.67052</td>
<td>0.56896</td>
<td>0.420293</td>
<td>3.24869</td>
<td>0.813833</td>
<td>1.0776</td>
<td>0.659329</td>
<td>0.468943</td>
<td>0.228985</td>
<td>3.24869</td>
</tr>
<tr>
<td>Scenario 3</td>
<td>0.978557</td>
<td>0.610363</td>
<td>0.67052</td>
<td>0.56896</td>
<td>0.420293</td>
<td>3.24869</td>
<td>0.813833</td>
<td>1.0776</td>
<td>0.659329</td>
<td>0.468943</td>
<td>0.228985</td>
<td>3.24869</td>
</tr>
<tr>
<td>Scenario 4</td>
<td>0.978557</td>
<td>0.610363</td>
<td>0.67052</td>
<td>0.56896</td>
<td>0.420293</td>
<td>3.24869</td>
<td>0.813833</td>
<td>1.0776</td>
<td>0.659329</td>
<td>0.468943</td>
<td>0.228985</td>
<td>3.24869</td>
</tr>
<tr>
<td>Scenario 5</td>
<td>0.978557</td>
<td>0.610363</td>
<td>0.67052</td>
<td>0.56896</td>
<td>0.420293</td>
<td>3.24869</td>
<td>0.813833</td>
<td>1.0776</td>
<td>0.659329</td>
<td>0.468943</td>
<td>0.228985</td>
<td>3.24869</td>
</tr>
</tbody>
</table>

Table 4(b) - The effect of higher flexibility on throughput (LQF Policy)

<table>
<thead>
<tr>
<th>Scenario</th>
<th>$\tau_1^R$</th>
<th>$\tau_2^R$</th>
<th>$\tau_3^R$</th>
<th>$\tau_4^R$</th>
<th>$\tau_5^R$</th>
<th>$\tau_s$</th>
<th>$\tau_1^P$</th>
<th>$\tau_2^P$</th>
<th>$\tau_3^P$</th>
<th>$\tau_4^P$</th>
<th>$\tau_5^P$</th>
<th>$\tau_s$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scenario 1</td>
<td>0.967742</td>
<td>0.524806</td>
<td>0.721104</td>
<td>0.590107</td>
<td>0.524955</td>
<td>3.32871</td>
<td>0.967742</td>
<td>1.02965</td>
<td>0.648367</td>
<td>0.46144</td>
<td>0.221512</td>
<td>3.32872</td>
</tr>
<tr>
<td>Scenario 2</td>
<td>0.982466</td>
<td>0.613041</td>
<td>0.654076</td>
<td>0.547927</td>
<td>0.414268</td>
<td>3.21178</td>
<td>0.749</td>
<td>1.07363</td>
<td>0.689526</td>
<td>0.470339</td>
<td>0.229287</td>
<td>3.21178</td>
</tr>
<tr>
<td>Scenario 3</td>
<td>0.982466</td>
<td>0.613041</td>
<td>0.654076</td>
<td>0.547927</td>
<td>0.414268</td>
<td>3.21178</td>
<td>0.749</td>
<td>1.07363</td>
<td>0.689526</td>
<td>0.470339</td>
<td>0.229287</td>
<td>3.21178</td>
</tr>
<tr>
<td>Scenario 4</td>
<td>0.982466</td>
<td>0.613041</td>
<td>0.654076</td>
<td>0.547927</td>
<td>0.414268</td>
<td>3.21178</td>
<td>0.749</td>
<td>1.07363</td>
<td>0.689526</td>
<td>0.470339</td>
<td>0.229287</td>
<td>3.21178</td>
</tr>
<tr>
<td>Scenario 5</td>
<td>0.982466</td>
<td>0.613041</td>
<td>0.654076</td>
<td>0.547927</td>
<td>0.414268</td>
<td>3.21178</td>
<td>0.749</td>
<td>1.07363</td>
<td>0.689526</td>
<td>0.470339</td>
<td>0.229287</td>
<td>3.21178</td>
</tr>
</tbody>
</table>
Figure 1 – An example of a flexible queueing system

$\lambda_i$: arrival rate of customer $P_i$

$\mu_i$: service rate of server $R_i$
Flexible Queuing System

\[ G(V,E) \]

\[ V_1 \]
\[ V_2 \]

State Vector \( N = \{n_1, n_2, n_3, n_4\} \)

Maximal Matching 1

\( n_1 = q_1 + s_4, n_2 = q_2 + s_1, n_3 = q_3 + s_2, n_4 = s_1 \)

Maximal Matching 2

\( n_1 = q_1 + s_1, n_2 = q_2 + s_3, n_3 = q_3 + s_2, n_4 = s_4 \)

Maximal Matching 3

\( n_1 = q_1 + s_1, n_2 = q_2 + s_2, n_3 = q_3 + s_3, n_4 = s_4 \)

Maximal Matching 4

\( n_1 = q_1 + s_1, n_2 = q_2 + s_2, n_3 = q_3 + s_3, n_4 = s_4 \)

State Vector \( N = \{n_1, n_2, n_3, n_4\} \)

Figure 2 – Example Maximal Matchings
Figure 3 – Flexibility scenarios for Observation 1
Figure 4 – Flexibility scenarios for Observation 2
Figure 5 - The effect of chaining on system throughput

Figure 6 – The effect of flexibility on system throughput
Figure 7 – Increasing flexibility one arc at a time
Figure 8 - Flexibility allocation scenarios for Observation 3

Figure 10 - Flexibility allocation scenarios for observation 4
Figure 9 - The effect of flexibility asymmetry on system throughput

Figure 11 – The effect of demand asymmetry
Figure 12 - The effect of demand on performance of queue selection policies