A QUEUEING NETWORK MODEL OF A
CIRCUIT SWITCHING ACCESS SCHEME
IN AN INTEGRATED SERVICES ENVIRONMENT

by

G.Y. Fletcher
H.G. Perros
W.J. Stewart

Center for Communications and Signal Processing

Computer Science Department
North Carolina State University
Raleigh, N.C. 27695-7914

CCSP-TR- 84/9

September 1984

Abstract:

A closed cyclic queueing system with multiple classes is analyzed numerically. The queueing system was originally motivated by a need to model circuit switching traffic in a data multiplexing scheme which provides both circuit and switching capabilities. The queueing system consists of two nodes, namely the primary and the secondary node. Customers in the primary node may require a random number of servers simultaneously. The secondary node simply provides the means of modelling the arrival process to the primary node. An efficient numerical procedure was devised with a view to obtaining performance measures such as throughput, queue-length distribution and utilization.

1. Introduction

Multi-server queueing systems in which a customer requires a random number of servers simultaneously, arise naturally in many telecommunication and computer systems. For instance, such queueing systems arise in satellite communication systems operating under a frequency division multiple access scheme (FDMA) or under a time division multiple access scheme (TDMA). In such schemes, a customer cannot start its service until all the requested servers (bandwidths in the case of FDMA or time slots in the case of TDMA) are available. Upon completion of its transmission all the servers are released simultaneously.

It is interesting to note that relatively little has been reported in the literature on the analysis of this class of queueing systems, despite the fact that they commonly arise in many real-life situations. Green [7] studied an M/M/s version of this queueing system assuming that the servers assigned to the same customer do not end service simultaneously. Thus, these servers become available independently. Analytic expressions for the distribution of the waiting time in the queue and the distribution of the number of busy servers were obtained. The same model was studied by Federgruen and Green [4] assuming that each server has a general service time distribution. The queue-length distribution was obtained approximately. Brill and Green [2] obtained the waiting time distribution of a customer in a similar system assuming that all servers associated with a customer end service at the same time. The derivation of this distribution is obtained using the system point approach (see Brill [3]). Explicit solutions were obtained for the case of a two-server queueing system. A comparison of various service disciplines associated with this class of queueing systems can be found in Green [8].

Arthurs and Kaufman [1] studied an Erlang loss type of system assuming that customers require simultaneous service from a random number of servers. Servers assigned to the same customer begin and end service concurrently. The steady-state distribution of the number of customers in the system is shown to have a product-form solution and is dependent on the service time distributions only

through their means. This product-form solution also holds for general service distributions (see Kaufman [9] and Schwartz and Kraimeche [11]). Kim [10] analyzed numerically a system similar to the one studied by Brill and Green [2] by devising an algorithmic approach to obtain the expected number of customers in the system, when the system size is not too large. Finally, Gopal and Stern [6] studied the problem of controlling access to a system similar to the one studied by Arthurs and Kaufman [1] with a heterogeneous mix of circuit-switched traffic.

In this paper, we analyze numerically a closed cyclic queueing system with multiple classes consisting of two nodes, namely the primary node and the secondary node. The primary node consists of M identical servers fed by a single queue. A class r customer requires r servers simultaneously, which are all released upon service completion. The service time of a customer is exponentially distributed with a class dependent mean. Customers departing from the primary node enter the secondary node. The secondary node simply provides the means of modelling the arrival process to the primary node. In particular, it can be employed to model a state-dependent or a state-independent arrival process.

The queueing system studied here was originally motivated by a need to model data multiplexing schemes for local networks, which provide both circuit switching and packet switching capabilities. In such schemes, transmission over the digital channel is organized into time slots. Successive slots are grouped into frames which are identical in structure and which occur consecutively. Let s be the number of slots per frame. Each frame is segmented into two compartments, namely the circuit switching compartment and the packet switching compartment. The circuit switching compartment is not permitted to use more than n_1 ($n_1 < s$) slots per frame. For a given frame, unused circuit switching slots are allocated to the packet switching compartment. A circuit switching customer, typically, may request one or more slots per frame. These slots are used by the

same customer for as many successive frames as required. Upon completion of its transmission, the customer releases all his slots simultaneously.

The primary node of the queueing system studied in this paper, can be used to model the circuit switching compartment of such an integrated data multiplexing scheme. In particular, each circuit switching slot can be represented by a server. Thus, the primary node will consist by as many servers as the maximum number of slots per frame allocated to circuit switching traffic. The secondary node can be used to model the arrival process of circuit switching customers. In particular, the following two real-life situations can be easily modelled.

- a) There is an infinite population of circuit switching customers. An arriving customer that cannot start transmission immediately is forced to wait in a queue. Upon completion of its transmission, the customer departs from the system.

The queueing model can be used to study this system by assuming that the secondary node consists of one server whose mean service time is equal to the mean inter-arrival time of circuit switching customers. A customer, upon arrival at the secondary node, is forced to wait if the server is busy. The total number of customers in the queueing system is assumed to be large enough so that the probability the secondary node is empty is negligible.

- b) There is a finite population of circuit switching customers. A customer that cannot start transmission immediately is forced to wait in a queue. Customers that are not queueing up or transmitting are assumed to be in a "think" state. This system is analogous to the well-known time-sharing (or machine interference model).

The queueing system studied in this paper will reflect the above system if the secondary node is modelled as an infinite server queue.

The queueing system studied here is an extension of an earlier model considered by Fletcher, Perros, and Stewart [5]. This earlier model consisted of a closed single node queueing system with multiple classes. The single node consisted of M identical servers fed by a single queue, and it is identical to the primary node of the queueing system studied in this paper. However, no secondary node was considered. A customer, upon service completion, was simply fed back to the end of the queue. An efficient numerical procedure was developed to analyze this closed single node queueing system, with a view to obtaining performance measures such as throughput, queue-length distribution, and the distribution of the number of busy servers. The model analyzed in this paper can be used to reproduce this earlier model simply by allowing the mean service time at the secondary station to tend to zero.

In the following section we describe the queueing model and outline some underlying considerations for the numerical solution. In section three, a state description for the model is introduced and the generation of the transition rate matrix is discussed in detail. Sections four and five describe the block Gauss-Seidel method of solution and implementation considerations respectively. Finally, some numerical results are presented in section six.

2. The queueing model

As mentioned in the introduction, the model under investigation is a two node closed cyclic queueing system. The primary node consists of M identical servers fed by a single queue. The secondary node, which provides a mechanism to model the arrival process is modelled as an infinite server queue. Schematically, this may be represented as follows

Figure 2.1. The queueing model under investigation.

Customers are chosen from the primary queue by means of the First-Come-First-Served (FCFS) scheduling discipline. Each customer can request more than one server; in fact, a customer may require any number of servers up to the maximum of M . The customer at the head of the queue remains in the queue until the total number of servers he requires becomes available at which time he is simultaneously allocated

this number. Since all the servers are identical, the particular ones allocated is irrelevant. Note that the FCFS discipline may force customers (other than the first) to wait even though there may be sufficient resource available (and hence idle) to service their requests. The advantage of the FCFS algorithm is that it prevents the undesirable phenomenon of "indefinite postponement". Note also that FCFS may result in several customers being able to enter service simultaneously upon completion of a service. Each customer monopolizes all the servers that have been allocated to him for a period of time that is exponentially distributed. At the end of this period, the customer simultaneously releases all this resource. For the purposes of the model, we shall assume that a customer belongs to one of R possible classes ($R \leq M$). A class r customer ($1 \leq r \leq R$) requires r servers simultaneously. To further generalize the model, we shall also assume that the exponential service time distribution may be class dependent.

At the secondary node, there is no concept of class. Each customer receives service which is exponentially distributed with rate λ . Naturally, because of the ample server property, no queueing takes place. On exiting from the secondary server, a customer proceeds to the primary server, but does not specify his class until he reaches the head of the queue.

There are no known analytical methods available for solving such a queueing system. We shall use numerical techniques to obtain a solution since the only other alternative, simulation, would be much too expensive. The numerical approach involves choosing an appropriate state description vector for the system; determining the corresponding transition rate matrix, i.e. the matrix whose (i,j) component denotes the rate of transition from state i to state j , and finally calculating from this matrix the stationary probability vector of the system, i.e. the vector whose i th component denotes the long run probability of being in state i . From this vector we can easily derive all the parameters of interest: the

throughput (i.e. the number of customers served per unit time), the queue-length distribution and the distribution of the number of busy (or idle) servers. We shall now outline the numerical procedure in somewhat more detail.

It is often possible to represent the behavior of a queueing system by describing all the different states which it may occupy and by indicating how it moves from one state to another in time. If the time spent in any state is exponentially distributed, or, in other words, if the probabilities of transition depend only on the state currently occupied by the system and not on any previously occupied state, then the system may be represented by a Markov chain. Even when the system does not possess this exponential property explicitly, it is usually possible to construct a corresponding implicit Markov representation.

Consider a system which is modelled by a continuous time Markov chain. Let $\xi_i(t)$ be the probability that the system is in state i at time t .

$$\text{Then } \xi_i(t+dt) = \xi_i(t)(1 - \sum_{j \neq i}^n \alpha_{ij} dt) + (\sum_{k \neq i}^n \alpha_{ki} \xi_k(t))dt + o(dt),$$

where α_{ij} is the rate of transition from state i to state j , and n is the total number of states.

$$\text{Define } \alpha_{ii} = - \sum_{j \neq i}^n \alpha_{ij}.$$

$$\text{Then } \xi_i(t+dt) = \xi_i(t) + (\sum_{k=1}^n \alpha_{ki} \xi_k(t))dt + o(dt)$$

$$\text{and } \lim_{dt \rightarrow 0} [\xi_i(t+dt) - \xi_i(t)]/dt = \lim_{dt \rightarrow 0} (\sum_{k=1}^n \alpha_{ki} \xi_k(t) + o(dt)/dt),$$

$$\text{i.e. } \dot{\xi}_i(t) = \sum_{k=1}^n \alpha_{ki} \xi_k(t)$$

In matrix notation

$$\dot{x}(t) = A^T x(t)$$

where $x(t)$ is a vector of length n whose i th component is $x_i(t)$ and A is a matrix of order n whose (i,j) element is a_{ij} .

At steady state, the rate of change of $x(t)$ is zero, and therefore

$$A^T x = 0 \quad (1)$$

The vector x is the required stationary probability vector (x_i is the probability of being in state i at statistical equilibrium), and may be obtained by applying equation solving techniques to the homogenous system of equations (1).

Note that $A^T x = 0 \Rightarrow A^T \Delta t x + x = x$

i.e. $P^T x = x$, where $P^T = A^T \Delta t + I$

If Δt is chosen such that $\Delta t \leq (\max_i |a_{ij}|)$, then the matrix P is a stochastic matrix. It may be regarded as the transition probability matrix for a discrete time Markov system in which transitions take place at intervals of Δt , Δt being sufficiently small to ensure that the possibility of two changes of state within this interval is negligible.

We note that the matrix Q defined as $Q = -A^T$ is a singular irreducible M-matrix with zero column sums. Such a matrix is sometimes called a "Q-matrix". The matrix A is called the infinitesimal generator or the transition rate matrix. The system of equations, (1), is sometimes referred to as the stationary form of the Chapman-Kolmogorov equations, but more often simply as the global balance equations. Note that since this system of equations is homogeneous, a zero pivot will occur during the last decomposition stage of Gaussian elimination, (See Stewart [12]). However, we have not used the fact that the sum of the probabilities must equal one, (i.e. our normalizing equation). Eliminating one of the equations from $A^T x = 0$ and replacing it with the normalizing equation yields a non-homogeneous system which indeed may be solved without pivoting, (See Stewart [12]). Alternatively, $A^T x = 0$ can be solved directly by Gaussian elimination without pivoting since the zero pivot must occur last.

Problems arise from the computational point of view because of the extremely large number of states which many systems may occupy. As will be shown below, it is not uncommon for hundreds of thousands of states to be generated, even for simple applications. Since the matrices involved are large and extremely sparse, numerical iterative methods such as Gauss-Seidel (either point or block) are usually recommended (unless, as sometimes happens, the non-zero structure of the matrix is highly regular in which case it may be advantageous to employ a direct method).

3. Generation of the Transition Rate Matrix

If we assume that there are R distinct customer classes, then we may describe the state of our system by the vector

$$(n, c_1, c_2, \dots, c_r, \dots, c_R)$$

of length $R+1$. Here n denotes the number of customers waiting in the secondary node, and c_r denotes the number of customers of class r in service in the primary node.

Computer generation of the transition rate matrix requires the solution of three basic problems: The first, is simply to enumerate all of the states of the system. The second, is to associate the proper enumeration integer with the vector description of any arbitrary state. The third, is to find all of the states which are accessible from an arbitrary given state (as well as the rate associated with each transition). These we have termed the "enumeration" problem, the "translation" problem, and the "transition" problem respectively.

An enumeration is simply a listing of all possible vector state descriptions. The ordinal position of each state vector in an enumeration is called its state number. Now, all states with identical n parameter, i.e. all states with the same number of customers in the secondary node, are regarded as forming a block. States within a given block are enumerated using the algorithm first reported in an earlier paper (see Fletcher, Perros, and Stewart [5]), where we studied a single node version of the queueing system considered here. In particular, states within a block are enumerated in a somewhat natural progression from those states where the number of customers in service is high to those where it is low. Finally, the various blocks of states are arranged in a decreasing order of n . This has the effect of enumerating the smaller blocks first, since having a large number of customers in the secondary node means that fewer are in the primary one.

The translation algorithm produces the correct state number for a given state vector. Due to the frequency with which it is called, it is perhaps the most time critical routine in the matrix generation procedure. The translation algorithm is implemented via a binary "dictionary" tree. The components of the state vector provide keys describing a path to the leaf which contains the associated state number. This algorithm was first reported in [5]. Each block of states is maintained in a separate tree. Thus, there are as many trees as blocks of states. For each state vector, parameter n furnishes a key describing which tree contains the state number.

The transition algorithm finds all the states which are accessible from an arbitrary given state as well as their corresponding transition probabilities. A state transition may be initiated by a departure from the primary or secondary node. Transitions due to a departure from the primary node are the most difficult to handle. This is because a departing customer may free up several servers, thus giving rise to different feasible combinations of customers that may enter service. Each of these combination of customers may correspond to a different state. The transition algorithm is primarily concerned with finding all these states. The key idea in this algorithm is the use of the displacement table. We digress for a moment to explain this idea.

We recall that a customer in the primary node does not specify its class until it reaches the head of the queue. Furthermore, the customer at the head of the queue (hereafter referred to as HQ) has a class with a service requirement exceeding the current number of idle servers. The remaining customers in the queue may be of any class. Now, upon departure of a customer from the primary node, the total number of idle servers is increased. The new idle resource may exceed the HQ requirement, in which case the HQ will enter service. Now, the new HQ may be of any class. We call the system configuration at this point as being in a base state. The accessible states due to this departure from the

primary node, can be easily determined using this base state. In particular, they can be determined by examining the various combinations of customers (waiting in the queue) that can enter service. The vector description of each of the resulting states may be regarded as a displacement of the base state vector. In fact, the set of possibilities for how many of each customer class may enter service can be described numerically by a set of vector displacements. The crucial observation here is that this set of displacements depend only on the excess service resource available and the length of the queue. It does not depend on the base state! Our algorithm capitalizes on this fact by maintaining a displacement table in memory which it consults to facilitate the generation of state transitions. Each transition represented in the displacement table has an associated rate which is also stored in the table.

The operation of the transition algorithm may be summarized as follows.

```

if (secondary node is nonempty) then
  assume departure from secondary node
  if (queue is nonempty) then
    customer enters queue **
  else
    for (each customer class) do
      if (service requirement exceeds available resource) then
        customer waits in queue **
      else
        customer enters service **
if (primary node is nonempty) then
  for (each customer class in service) do
    customer departs service enters secondary node
    if (HOQ requirement exceeds current available resource) then
      queue is unchanged **
    else
      HOQ will enter service
      for (each entry in the displacement table consistent with
        available resource and queue length) do
        HOQ and (perhaps) other customers enter from queue **
end

```

** indicates that

- A new accessible state has been reached.
- The state description must be translated into a state number.
- The associated transition rate must be recorded.

The actual implementation of the algorithms requires the use of many control variables to keep all of the necessary information current as well as the data structures required for the dictionary trees and the displacement table. We sketch the procedure for generating the matrix in terms of the algorithms which we have discussed.

For a given set of system parameters (# customers, # servers, # classes, etc.) the enumeration algorithm generates the vector description of each state (matrix row) which is possible.

For each of these possible states the transition algorithm generates the vector description of every accessible state together with the associated transition rate.

The state number (matrix column) for each accessible state is provided by the translation algorithm allowing the transition rate to be placed in the correct matrix entry.

An examination of the transition process will reveal that all of the states accessible from a given state will either be in the preceding block (in the case of a departure from the primary node) or in the succeeding block (in the case of a departure from the secondary node). Thus the rate transition matrices which we generate will all have a block tridiagonal structure. Our solution technique has been chosen to exploit this structure.

4. The Numerical Approach

When the states of the system are arranged according to the order prescribed in the previous section, the transition rate matrix will possess a block tridiagonal form. Some examples obtained from a number of different cases are presented in figures 4.1 through 4.4 below.

Figure 4.1. An example of a transition rate matrix.

Figure 4.2. An example of a transition rate matrix.

Figure 4.3. An example of a transition rate matrix.

Figure 4.4. An example of a transition rate matrix.

We shall use the following block notation to represent these matrices.

$$A = \begin{bmatrix} A_{00} & A_{01} & & & & \\ A_{10} & A_{11} & A_{12} & & & \\ A_{21} & A_{22} & A_{23} & \ddots & & \\ \vdots & \vdots & \vdots & \ddots & \ddots & \\ \text{(Zero)} & & & & A_{N-1,N-2} & A_{N-1,N-1} & A_{N-1,N} \\ & & & & A_{N,N-1} & A_{NN} & \end{bmatrix}$$

The number of blocks, $N+1$, is obviously related to the number of customers N that circulate in the network; block j is associated with states in which j customers are at the primary node. If n_j is the number of such states, then the order of the transition rate matrix A is given by $n = \sum_{j=0}^N n_j$. Notice that the diagonal blocks are themselves diagonal submatrices. As we shall see below, this, together with the fact that the diagonal element in each row is equal to the negated sum of the off diagonal elements in that row, makes the block Gauss-Seidel method extremely attractive as a means of solution.

We shall assume that the stationary probability vector x and successive iterates $x^{(s)}$ to the solution vector are partitioned conformally with A :

$$\text{i.e. } x^T = (x_0^T, x_1^T, \dots, x_N^T) \in \mathbb{R}^n$$

and

$$x_j \in \mathbb{R}^{n_j} \text{ with } \sum_{j=0}^N n_j = n.$$

For block tridiagonal matrices, the block Gauss-Seidel iteration method may be written algorithmically as follows

For $s = 0, 1, 2, \dots$ (until convergence)

For $j = 0, 1, 2, \dots, N$

$$\text{Solve } A_{jj}x_j^{(s+1)} = -A_{jj-1}x_{j-1}^{(s+1)} - A_{jj+1}x_{j+1}^{(s)} \quad (4.1)$$

where it is implicitly assumed that the matrices A_{jj-1} and A_{jj+1} are identically zero when $j=0$ and $j=N$ respectively.

Since the diagonal blocks of A are known to be nonsingular and are diagonal, their inverses, A_{jj}^{-1} $j=0, 1, \dots, N$, may be trivially computed and hence equation (4.1) may be written as

$$x_j^{(s+1)} = -A_{jj}^{-1}A_{jj-1}x_{j-1}^{(s+1)} - A_{jj}^{-1}A_{jj+1}x_{j+1}^{(s)}$$

i.e.
$$x_j^{(s+1)} = V_jx_{j-1}^{(s+1)} + W_jx_{j+1}^{(s)}$$

Note also that V_j and W_j may be computed before the iteration procedure is initiated, so that it is not necessary to store the diagonal elements explicitly during the actual computation of the stationary probability vector.

Schematically, the matrix is then as follows

$$\begin{bmatrix} 0 & V_0 & & & \\ W_1 & 0 & & & \\ & W_2 & 0 & V_2 & \\ & & \ddots & \ddots & \ddots \\ & & & W_n & 0 & V_{N-1} \end{bmatrix}$$

or, in transposed form

$$\begin{bmatrix} 0 & w_1^T & & & \\ v_0^T & 0 & w_2^T & & \\ & v_1^T & 0 & w_3^T & \\ & & \ddots & \ddots & \ddots \\ & & & v_{N-2}^T & 0 & w_N^T \\ & & & & v_{N-1}^T & 0 \end{bmatrix}.$$

Since the iteration vector $x^{(s)}$ is similarly partitioned, the effect of performing an iteration may be viewed as the product of

$$\begin{bmatrix} 0 & w_1^T & & & \\ v_0^T & 0 & w_2^T & & \\ & \ddots & \ddots & \ddots & \\ & & v_{N-1}^T & 0 & \end{bmatrix} \begin{bmatrix} x_0^{(\cdot)} \\ x_1^{(\cdot)} \\ \vdots \\ x_N^{(\cdot)} \end{bmatrix}$$

with the only difference being that as portions of the new iterate are formed they are used in all succeeding computations rather than the values of the previous iterate.

In the absence of any better information, it is suggested that the initial approximation be taken as

$$x^{(0)} = \left(\frac{1}{n}, \frac{1}{n}, \dots, \frac{1}{n} \right)^T.$$

5. A Sparse Storage Scheme and Other Implementation Considerations

We now turn to a discussion of the sparse storage scheme which is used to store the non-zero elements of the transition rate matrix. Given the size of the matrices involved, such schemes not only make sense from a space saving point of view, but are often prerequisite if a solution at all is to be obtained. Given the form of the matrix equations presented in the previous section, it should be noted that the only operations required of the elements is multiplication with a vector. Consequently, the storage scheme used should make such a product convenient. However, other considerations must also be accommodated; in particular, the manner in which the non-zero elements themselves are generated. Recall that the element a_{ij} of the transition rate matrix denotes the rate of transition from state i to state j and that the matrix generation procedure obtains all possible transitions from a given state before proceeding to determine transitions from the next state. As a result, the matrix A is generated by rows. However, the stationary probability vector x which we are seeking is the solution to $A^T x = 0$, which involves the transpose of the transition rate matrix.

We shall therefore store the non-zero elements by rows as they occur but when performing the multiplications during the iteration phase, we shall need to remember that it is the transpose that is used.

We shall describe the sparse storage scheme which we recommend by means of the following example, which has the structure of the upper left hand portion of a typical transition rate matrix for this application. The diagonal elements have not been written explicitly.

of row $k-1$ (viz. $\alpha_{k-1,j}$). The order of elements within each row is immaterial. The integer array IV will denote first the number of non-zero elements in a given row and then the column position of each of these elements. Obviously this latter must correspond to the ordering in V. Thus, for example, one possible ordering is

$$\begin{array}{l} V \quad [\alpha_{02}\alpha_{01} \mid \alpha_{13}\alpha_{14} \mid \alpha_{24}\alpha_{25} \mid \alpha_{36}\alpha_{37} \mid \alpha_{14,8}\alpha_{14,7} \mid \alpha_{59} \mid \dots] \\ IV \quad [2 \ 2 \ 1 \mid 2 \ 3 \ 4 \mid 2 \ 4 \ 5 \mid 2 \ 6 \ 7 \mid 2 \ 8 \ 7 \mid 1 \ 9 \mid \dots] \end{array}$$

A similar arrangement holds for the non-zero elements in sub diagonal blocks.

As the non-zero elements are being generated and inserted into the arrays V and W, it is possible to incorporate the effect of the multiplication by $-A_{jj}^{-1}$ simultaneously. The effect of the multiplication is to multiply all of the elements in a given column, column k say, by $-\frac{1}{\alpha_{kk}}$ where $\alpha_{kk} = -\sum_{i=1}^n \alpha_{ik}$. (Note column i , not row i , since we require the transpose of the matrix). For the elements in subdiagonal blocks, this multiplication may be performed as soon as the elements themselves are determined and the product may be stored directly into W. This is because the required diagonal element will always have been previously generated. This is not true for non-zero elements in the super diagonal blocks so that it will always be necessary to backtrack a little. For example α_{12} and α_{13} must be multiplied by α_{22} and α_{33} respectively, but neither α_{22} nor α_{33} will be available until the second block has been completed..

It now only remains to show how the product $W_{j+1}x_{j+1}$ (or $V_{j-1}x_{j-1}$) is to be carried out given that W_{j+1} (V_{j-1}) is stored in this compact form. Once again we shall illustrate the procedure by means of an example.

We have $x_1^{(s+1)} = v_0^T x_0^{(s+1)} + w_2^T x_2^{(s)}$ where

$$v_0 = [v_{01} \ v_{02}]; \quad w_2 = \begin{bmatrix} w_{31} & & & \\ w_{41} & w_{42} & & \\ & & w_{52} & \end{bmatrix}$$

$$x_0 = [\xi_0]; \quad x_1 = \begin{bmatrix} \xi_1 \\ \xi_2 \end{bmatrix}; \quad x_2 = \begin{bmatrix} \xi_3 \\ \xi_4 \\ \xi_5 \end{bmatrix},$$

v_0 is stored as

$$V: [v_{01}, v_{02}, \dots]$$

$$IV: [1, 1, 1, 1, \dots]$$

while w_2 is stored as

$$W: [\dots \mid w_{31} \ w_{41} \ w_{42} \ w_{52} \mid \dots]$$

$$IW: [\dots \mid 1 \ 1 \ 2 \ 1 \ 2 \ 1 \ 2 \mid \dots]$$

It may easily be verified that the required result is given by

$$\xi_1^{(s+1)} = w_{31} \xi_3^{(s)} + w_{41} \xi_4^{(s)} + v_{01} \xi_0^{(s+1)}$$

$$\xi_2^{(s+1)} = w_{41} \xi_4^{(s)} + w_{52} \xi_5^{(s)} + v_{02} \xi_0^{(s+1)}$$

One possible procedure is to first zero out the components of $x_1^{(s+1)}$; i.e.

$\xi_1^{(s+1)} = \xi_2^{(s+1)} = 0$, and to add parts of the result into the appropriate position as and when they are obtained. Consider, for example, how $w_2^T x_2^{(s)}$ is formed and added into $x_1^{(s+1)}$. The elements in the array W are processed one at a time. For each element in this array we must determine the appropriate element in $x_2^{(s)}$ with which to form the product. However this merely increases by one for each column/

row processed. Thus ω_{31} is multiplied by ϵ_3 , the next two elements ω_{41} and ω_{42} are both multiplied by ϵ_4 and finally ω_{52} is multiplied by ϵ_5 . Thus we see that both the arrays W and x are processed linearly. Once each of these multiplications have been performed, the result must be added into the appropriate position of $x_1^{(s+1)}$. Again the determination of the correct position is easily obtained by progressing linearly through the array IW (remembering that the actual position indicators are preceded by the number of elements. Thus

$\omega_{31}\epsilon_3$ is added into position 1

$\omega_{41}\epsilon_4$ is added into position 1

$\omega_{42}\epsilon_4$ is added into position 2

$\omega_{52}\epsilon_4$ is added into position 2

The product $V_0 x_0^{(s+1)}$ may be computed and added into $x_1^{(s+1)}$ in a similar fashion.

Thus we see that although the matrix is not stored in the most obvious fashion for multiplication with a vector, such computations may still be efficiently carried out.

6. Numerical Results

The numerical procedure outlined in the previous sections was implemented on a VAX 11/780. Numerical experience with this procedure showed that most of the processing time was devoted to the matrix generation procedure. The calculation of the stationary probability vector required only a small fraction of the total processing time. Complicated configurations requiring more than two classes of customers and a large number of servers were easily analyzed. Time complexity limitations became apparent only when there were R classes present, where R is greater than 10. In such a case, one can still use the algorithm effectively by reducing the number of classes to a more manageable level. Such a reduction can be achieved by appropriately lumping classes of little operational interest into a single class.

It has been observed empirically that the blocks on the lower diagonal in the transition rate matrix are all identical after a certain block. In particular, they are all identical after the block number $\lfloor M/r_0 \rfloor + 1$, where M is the number of servers in the primary node, and r_0 is the smallest permissible class ($r_0 \geq 1$). In view of this, one may not need to generate all the lower diagonal blocks. For example, in figure 4.1 it suffices to generate the first two lower diagonal blocks, seeing that all the remaining blocks are mere repetitions of the second block. This particular behaviour can lead to substantial savings in processing time when generating the rate matrix. Obviously, in order to avail of this saving the number of the lower diagonal blocks that have to be generated should be greater than $\lfloor M/r_0 \rfloor + 1$.

The numerical procedure was employed to obtain various measures of performance as shown in figures 6.1 to 6.6. The number of servers in the primary node was fixed to 15. A customer in the primary node may require 1, 4, or 7 servers (i.e. it may be of class 1, 4 or 7) with probability 0.7, 0.2, and 0.1 respectively. The remaining parameters which characterize the queueing network

under study were varied in order to obtain the results shown in the figures below. These parameters are: a) number of customers in the system; b) the vector (μ_1, μ_4, μ_7) of service rates of customers class 1, 4, and 7 respectively; and c) the mean service time at the secondary node $1/\lambda$. The performance measures obtained are related to the primary node. These are the throughput, the mean queueing time and the mean number of busy servers. The throughput is defined as the average number of service completions per unit time.

Figure 6.1 gives the throughput of the primary node as a function of the number of customers in the system plotted for various values of (μ_1, μ_4, μ_7) . Figure 6.2 gives similar information for the throughput, but as a function of $1/\lambda$. The mean queueing time in the primary node is given in Figure 6.3, as a function of the number of customers in the system plotted for various values of (μ_1, μ_4, μ_7) . Figure 6.4 gives similar information for the mean queueing time, but as a function of $1/\lambda$. Figure 6.5 gives the mean number of busy servers at the primary node as a function of the number of customers, plotted for various values of $1/\lambda$. The same information, but slightly rearranged, is given in Figure 6.6.

Acknowledgement

We would like to thank Mr. Yun-Cheng Liu, Center for Communications and Signal Processing, N. C State Univer., for the excellent programming support.

Figure 6.1: Throughput of the primary node vs. the number of customers in the system, plotted for various values of (μ_1, μ_4, μ_7) ; $1/\lambda=0.1$.

Figure 6.2: Throughput of the primary node vs. mean service time at the secondary node, plotted for various values of (μ_1, μ_4, μ_7) ; the number of customers in the system = 20.

Figure 6.3: Mean queueing time in the primary node vs. the number of customers in the system, plotted for various values of (μ_1, μ_4, μ_7) ; $1/\lambda=0.1$.

Figure 6.4: Mean queueing time in the primary node vs. mean service time at the secondary node, plotted for various values of (μ_1, μ_4, μ_7) ; the number of customers in the system = 20.

Figure 6.5: Mean number of busy servers at the primary node vs. the number of customers in the system, plotted for various values of $1/\lambda$; $(\mu_1, \mu_4, \mu_7) = (1, 2, 4)$.

Figure 6.6: Mean number of busy servers at the primary node vs. mean service time at the secondary node, plotted the various values of the number of customers in the system; $(\mu_1, \mu_4, \mu_7) = (1, 2, 4)$.

References

1. Arthurs, E., and Kaufman, J. S., "Sizing a message store subject to blocking criteria", in Performance of Computer Systems, M. Arato, A. Butrimenko, and E. Gelenbe, (Eds.), Amsterdam, The Netherlands, North Holland, (1979) 547-564.
2. Brill, P. H., and Green, L., "Queue in which customers receive simultaneous service from a random number of servers - a system point approach", *Mgmt. Sci.*, 30 (1984) 51-68.
3. Brill, P. H., "An embedded level crossing technique for dams and queues", *J. of Appl. Prob.*, 16 (1979) 174-186.
4. Federgruen, A., and Green, L., "An M/G/c Queue in which the number of servers required is random", Columbia Business School Research paper 504A, Columbia University, (1983).
5. Fletcher, G. Y., Perros, H. G., and Stewart, W. J., "A queueing system where customers require a random number of servers simultaneously", CCSP-8412, Computer Science Dept., N. C. State Univ., (1984).
6. Gopal, I. S. and Stern, T. E., "Optimal call blocking policies in an integrated services environment", IBM Research Report RC 9825.
7. Green, L., "A queueing system in which customers require a random number of servers", *Oper. Res.*, 28 (1980) 1335-1346.
8. Green, L., "Comparing operating characteristics of queues in which customers require a random number of servers", *Mgmt. Sci.*, 27 (1981) 65-74.
9. Kaufman, J., "Blocking in a shared resource environment", *IEEE Trans. Comm.*, Com-29 (1981) 1474-1481.
10. Kim, S., "M/M/s queueing system where customers demand multiple server use", Ph.D. dissertation, Southern Methodist Univ., (1979).
11. Schwartz, M., and Kraimeche, B., "An analytic control model for an integrated node", Inforcom '83 San Diego, (1983).
12. Stewart, W. J., "A comparison of numerical techniques in Markov Modelling", *Comm. ACM*, 21 (1978) 144-151.