DESIGN AUTOMATION AND QUEUEING NETWORKS: AN INTERACTIVE SYSTEM FOR THE EVALUATION OF COMPUTER QUEUEING MODELS*

by

K. M. Chandy, T. W. Keller, J. C. Browne

This work was supported by NSF grant GJ-1084.

May 1972 TR-3

Technical Report No. 3

Department of Computer Sciences
The University of Texas
Austin, Texas 78712

DESIGN AUTOMATION AND QUEUEING NETWORKS: AN INTERACTIVE SYSTEM FOR THE EVALUATION OF COMPUTER QUEUEING MODELS*

K. M. Chandy
T. W. Keller
J. C. Browne

Department of Computer Sciences
The University of Texas at Austin

Abstract

Design automation techniques have been successfully used in wiring board layouts, circuit design analysis, and other areas. Simulation techniques have been used by system analysts to evaluate complex computer systems. In the early stages of a computer system design a designer will find an interactive package, which gives him real-time solutions for queueing models of complex computer systems, extremely useful. In the first few passes at a design, detailed simulation studies are too expensive and too slow. A conversational package which evaluates arbitrary configurations and gives approximate results is preferable to a slow, expensive, though accurate simulation. This paper is concerned with the real-time analysis of complex queueing network models which have been extensively used in computer systems analysis. In this paper we present the theory, algorithms and some programs for a system which departs radically from previous attempts at computer design aids for queueing network analysis in two ways: firstly the design automation system proposed here will give both algebraic and numerical answers to queries put forward by the analyst. For instance, the analyst may want to find out an algebraic expression for the throughput of a network as a function of several parameters. Or the analyst may want the output in the form of a graph of throughput as a function of a parameter. The system discussed in this paper will satisfy both types of requests. Secondly, the system allows the analyst to evaluate arbitrary networks constructed from a set of specified "building blocks." The "building blocks" are quite general--they include queues with devices (servers), branches, joins, and so on. The analyst is allowed to use as many building blocks as he pleases, and he can interconnect these building blocks in any pattern he chooses. Thus the system is moderately flexible and allows the analyst greater freedom in choosing the models best suited for his system.

I. Queuing Models of Computer Systems

A computer system consists of several different kinds of resources. Jobs (customers) enter the system and queue up for resources. Computer system resources include central processing units, memory, etc. When a job finishes getting service from one resource it may join a queue for another resource. A computer system can be modeled as a network of interconnected queues.

We shall be concerned with three service disciplines: First Come First Served (FCFS), Processor Sharing (PS), and Last Come First Served Preemptive Resume (LCFSPR). We shall refer to the last two disciplines as distribution independent disciplines for reasons to be explained later.

The PS discipline is the limiting case of Round-Robin-Fixed Quantum (RRFQ) as the quantum and switch times get arbitrarily small. All jobs in the queue are assumed to be serviced simultaneously: if k jobs are being simultaneously processed by a single device, then each of the k

*The work reported here was supported in part by NSF Grant GJ - 1084, "Design and Analysis of Multi-Programmed Computer Operating Systems."

jobs is processed at 1/k times the rate at which a job having the device all to itself is processed. The LCFSPR discipline is a priority-preemptive discipline in which the priority of a job is the clock time at which it enters the queue. The job with the highest priority is serviced while lower priority jobs wait. The PS discipline is a very good approximation to RRFQ when the quantum size is small. Since RRFQ is widely used, the PS discipline is of significant practical interest.

This paper makes contributions in two areas: some new results in queueing theory and computer systems modeling are presented, and methods for automatically obtaining algebraic solutions for queues (ASQ) are discussed.

Results in queueing networks

1. Distribution independent disciplines

The steady-state probabilities, utilizations, throughput, etc. of a queueing network in which all service disciplines are distribution independent (that is, PS or LCFSPR) depends on the mean service times and is otherwise independent of the service distributions, provided the distributions have rational Laplace transforms. This result was shown by Chandy [3]. It was first conjectured by Baskett [2] for the PS discipline. The ASQ system will handle networks with distribution independent service disciplines.

2. Job Typing

Most queueing network models assume that all jobs in the network have common service distributions and branching probabilities. However, some resources may be used primarily by a given class of jobs while other resources may be used extensively by other classes of jobs. Furthermore, service distributions may vary from one type of job to another. The CPU burst times of an interactive job are generally different from those of a batch job. We present solutions for queues with different types of jobs, where each type has its own branching probabilities and service distributions. We shall permit different service distributions for different job types only in queues with distribution independent service disciplines. The solutions for queueing networks with job typing is a new result. The ASQ system does not currently handle job typing, but we plan to implement it in the future.

3. Auto Correlated Job Behavior

Most queueing models have assumed that the service times and branching probabilities of a job are independent of past behavior. However, many jobs exhibit a behavior pattern. For instance, a job might have a series of short CPU burst times followed by a series of long burst times. In this paper we present a model for program behavior which takes this autocorrelation into account. We show that in queueing networks with distribution-independent service disciplines, the steady-state probabilities are independent of autocorrelated behavior: the steady-state probabilities depend only on the mean value. This result is important because it suggests that throughput and utilizations for computer systems in which CPU burst times exhibit certain patterns, can be obtained by making the (false) assumption that successive burst times are independent, provided that the CPU is serviced in a RRFQ manner with a small quantum. The ASQ system will handle networks in which jobs exhibit this behavior pattern.

4. Local Balance

The concept of local balance was developed by Chandy [3]. Local balance allows for trivial analysis of apparently complex queueing systems. It will not be discussed here, a detailed discussion is presented in [3].

Closed Queueing Networks

Consider the queueing network shown in Fig. 1. In this network there always are a fixed number N of jobs cycling around the network. A job enters the network at the "origin," traverses the network, goes to the "destination" and is recycled back to the origin. The ASQ system only handles closed networks at this time. We plan to implement the analysis of "open" networks in the ASQ system in the future. In an open network jobs enter the network from external sources, and may leave the network and go to external sinks.

Let p_{ij} be the probability that a job which finishes getting service in queue i joins queue j next. Let PTERM(i) be the expected service time in queue i. We assume initially that all jobs have common service distributions and common branching probabilities. We shall also assume that

all service times are exponentially distributed. Both assumptions will be relaxed later.

Consider the queueing network shown in Fig. 1. The Branch TERM for queue i (BTERM(i)) may be defined as the probability that a job gets to queue i on its way from the origin to the destination. Thus BTERM(1) = p_{01} and BTERM(4) = $p_{01} \cdot p_{14} + p_{02} \cdot p_{24}$. The BTERMs may be defined more formally as any set of numbers such that BTERM(i) = $\sum_j$ BTERM(j)p_{ji}, all i. The Processing TERM for queue i or PTERM(i) is defined to be the expected service time in queue i. We assume initially that there is only one device serving queue i. Let $(n_1, .., n_i, .., n_M)$ be the state of the system in which there are n_i jobs in queue i, all i. The state $(n_1, .., n_i, .., n_M)$ is said to be feasible if $n_1 + .. + n_i + .. + n_M = N$, where N is the total number of jobs in the closed network, and $n_1, .., n_M \geq 0$. Jackson [6] showed that the steady-state probabilities are:

$$P(n_1, .., n_i, .., n_M) = \text{NORM} \prod_i \{\text{BTERM}(i).\text{PTERM}(i)\}^{n_i} \quad (1)$$

Chandy [3] obtained the same result using local balance. Since local balance is the key to understanding how several queueing models can be solved with the ASQ system, we will discuss it briefly. For more detail see [3]. We note that in the ASQ system, the transition probabilities p_{ij} and the PTERMs may be specified as polynomial functions of several variables. The steady-state probabilities, utilizations, throughputs are computed by ASQ (Section II).

The Concept of Local Balance

A transition can be made out of state $(n_1, .., n_M)$ due to several events. A job can finish service in queue j and move to queue k, for any j, k. In this case the system transits to state $(n_1, .., n_{j-1}, .., n_k + 1, .., n_M)$. Now let us focus attention on only one kind of transition: a transition due to a job leaving queue i. This transition can result in any one of the new states $(n_1, .., n_i - 1, .., n_k + 1, .., n_M)$, k = 1, .., M, k ≠ i.

We seek a functional form $R(n_1, .., n_M)$ which is the steady-state probability if $(n_1, .., n_M)$ is a feasible state. In other words:

$$P(n_1, .., n_M) = \begin{cases} R(n_1, .., n_M) \text{ if } (n_1, .., n_M) \text{ is feasible} \\ 0 \text{ otherwise} \end{cases} \quad (2)$$

Local Balance Lemma

A sufficient condition for $R(n_1, .., n_M)$ to be the functional form for steady-state probabilities $P(n_1.., n_M)$ is that the rate at which the system leaves state $(n_1, .., n_M)$ due to a job moving out of queue i is $R(n_1, .., n_{i-1}, n_i - 1, n_{i+1}, .., n_M)$. BTERM(i) (3)

The proof is in [3]. We shall refer to equation (3) as the ceteris paribus equation. The steady-state probabilities (1) satisfy the ceteris paribus equation since the rate at which the system transits out of $(n_1, .., n_M)$ due to a job moving out of queue i is $P(n_1, .., n_M) \cdot u_i$ where $u_i = 1/\text{PTERM}(i)$ is the service rate in queue i.

Non-exponential Distributions

A service time distribution with a rational Laplace transform may be represented as a network of interconnected exponential stages, where the time spent by a job in stage i is exponentially distributed with mean $1/\mu_i$. A job starts at an entry point, traverses the network and then finishes service when it reaches the exit point. The network may have several branches and a path that a job traces may vary with each service. We shall refer to distributions which can be represented as a network of exponential stages as an EN (Exponential Network) distribution. To describe the status of a job we now have to describe both the queue the job is in, and the stage(Fig. 5).

Consider a closed queueing network in which some of the queues indexed i have EN service distributions and PS disciplines; other queues indexed j have EN service distributions and LCFSPR disciplines; and the remaining queues indexed k have exponential service distributions and FCFS discipline. For ease of exposition we shall merely refer to three queues, i, j, k, though in reality there may be several of each type. A state for this network is a set of vectors (y_i, y_j, y_k), where:

$y_i = (x_{1i}, x_{2i}, .., x_{Qi})$ where x_{ri} is the number of jobs in stage r of queue i.

$y_j = (w_{ij}, w_{2j}, \ldots)$, where w_{sj} is the stage in which the sth job in the the stack is in. The first job in the stack is always processed, and as new jobs arrive, they are placed on top of the stack.

$y_k = n_k$ is just the number of jobs in the queue.

Let BTERM(r, i) be the probability that a job enters stage r of queue i. Note that BTERM(r, i) is the probability that queue i is entered multiplied by the conditional probability that stage r of queue i is entered given that queue i is entered. Let $n_i = x_{1i} + .. + x_{Qi}$. The functional form of the steady-state probabilities is :

$$R(y_i, y_j, y_k) = \text{NORM} \cdot \text{TERM}_i \cdot \text{TERM}_j \cdot \text{TERM}_k \qquad (4)$$

$$\text{TERM}_i = \frac{n_i!}{x_{1i}! \ldots x_{Qi}!} \prod_r \{\text{BTERM}(r, i) \cdot \text{PTERM}(r,i)\}^{x_{ri}}$$

where PTERM(r, i) is the expected time spent by a single job in stage r of queue i, when it is the only job in the queue.

$$\text{TERM}_j = \prod_s \{\text{BTERM}(w_{sj}) \cdot \text{PTERM}(w_{sj})\}$$

$$\text{TERM}_k = \{\text{BTERM}(k) \cdot \text{PTERM}(k)\}^{n_k}$$

It is easy to verify that this functional form satisfies the ceteris paribus equation. Using (4) we find that the steady-state probabilities are still given by equation (1). This allows the ASQ system to analyze this sort of network, with distribution independent disciplines and EN distributions.

Job Typing

Let there be T types of jobs labeled 1, .., t, .., T. Let the probability that a type t job joins queue b after finishing service on queue c be p_{tbc}. The different types of jobs may use varying amounts of system resources. Let the expected time spent by a type t job in stage r of queue i be PTERM(t, r, i), if it is the only job in the queue. Suppose there are m_t jobs of type t in the network at all times. Using the same notation as in the previous paragraph a state of the network is a set of vectors (y_i, y_j, y_k) where

$y_i = (x_{11i}, x_{21i}, \ldots, x_{T1i}, \ldots, x_{1Qi}, \ldots, x_{TQi})$ where x_{tri} is the number of jobs of type t in stage r of queue i. y_j is similarly defined. $y_k = (z_{1k}, z_{2k}, \ldots, z_{Bk})$ where z_{dk} is the type of the dth job in the queue. However, we shall make the assumption that in a queue with a FCFS discipline, all job types have a common exponential distribution.

In analogy to the previous case we have:

$$\text{TERM}_i = \frac{n_i!}{x_{11i}! \ldots x_{TQi}!} \prod_t \prod_r \{\text{BTERM}(t,r,i) \cdot \text{PTERM}_{(t,r,i)}\}^{x_{tri}}$$

$$\text{TERM}_j = \prod_t \prod_r \{\text{BTERM}(w_{tsj}) \cdot \text{PTERM}(w_{tsj})\}$$

$$\text{TERM}_k = \prod_d \{\text{BTERM}(z_{dk}) \cdot \text{PTERM}(z_{dk})\}$$

It is easy to verify that the ceteris paribus equation is satisfied. Note, however, that the local balance lemma will have to be rephrased slightly, to take into account the different kinds of jobs.

Autocorrelated Job Behavior

The CPU burst times for a given job frequently exhibit a pattern as shown in Fig. In this paper we present a model for the generation of CPU burst times, which is in some respects similar to [4] Denning's models for working set size. . A job is said to have several "modes," and a job is in a single mode for the duration of a CPU burst. At the end of a burst, a job may transit from its present mode to a new mode. Transitions between modes are assumed to be a discrete-state-discrete-time Markov process, with each mode corresponding to a state of the process.

Each mode of a program may have its own service distribution and branching probabilities. Thus a program may change its behavior with time, and its present behavior will depend in some way on its history.

Consider the mode-transition probability matrix and the service density functions shown in Fig. 6 . A possible sequence of modes and burst times are shown in Fig. 6 . Preliminary investigations at the University of Texas at Austin, Computation Center, indicate that some programs do behave in this manner.

It is easy to show using local balance, that for distrubution-independent service discip-

lines, the steady-state probabilities and utilizations are independent of the modes of a job, and depend only on its mean service time, provided the service distributions in each mode have rational Laplace transforms and the mode transition matrix is ergodic. Note that we have used a partially observable Markov process to describe CPU burst generation.

A typical job is now described by four parameters: its type t, the mode m of the job (different types will have different modes), the stage of the distribution (different modes will have different stages), and of course the queue that the job is in. We shall define BTERM (t,m,r,i) in the usual way: it is the probability that a type t job in mode m enters stage r of queue i on its passage from the origin to the destination. BTERM(t,m,r,i) can be computed readily. The steady-state probabilities are analogous to the earlier paragraph on job typing. Once again it is relatively simple to check that the ceteris paribus equation is satisfied. The ASQ system will handle certain kinds of autocorrelated behavior.

Queueing Network Models -- Summary

In all the networks analyzed here, the probability that there are n_i jobs of a given type, mode etc. in queue i is proportional to the n_ith power of the product of the probability that the given type of job gets to queue i and the expected service time of this type of job in queue i. We also see that local balance is a very powerful analytical tool. Several other interesting results on computer queueing models can be derived using local balance. These results will be incorporated in the ASQ system, as the system is developed and extended. Paucity of space prevents us from discussing these results here.

We shall now discuss the ASQ system in some detail. In the following discussion we define the throughput of a closed network to be the rate at which jobs enter the network at the origin, or leave it at the destination.

The ASQ System

The program ASQ (Algebraic Solutions for Queues) will currently accept an arbitrary closed network composed of any number of branches, joins and queues with devices. We will assume that the branching probabilities are the same for all jobs of all types, and are constant. We will also assume that the service distributions in all queues with FCFS discipline are exponential, and the same for all types of jobs, at all times. In queues with PS or LCFSPR disciplines, the service distributions may have EN distributions (i.e. distributions with rational Laplace transforms), and may exhibit autocorrelated behavior in the manner described in the previous section. It is assumed that there is no limit on the number of jobs in a queue. All jobs enter the network at the origin, travel through the network and go to the destination. A job can enter a given queue at most once on its passage from the origin to the destination. A queue may be serviced by more than one device. If so, the devices are assumed to be in parallel, although they may have different exponential service rate distributions. A queue and its devices are called a node. The label (name) of a node is the label of the queue. Devices do not possess labels.

Nodes are connected by branches and joins. If upon completion of service at node J a job can go to more than one node, then branches are drawn from J to those nodes. Associated with a branch from some node J to a node K is the probability p_{JK}, the probability that upon completion of service at J the job will enter K. Naturally, the sum of the branching probabilities from a node must be 1. If upon completion of service at node J a job can go only to node K, then a join (a branch with probability 1) is drawn from J to K. The triple (J,p_{JK}, K) is called an edge, and establishes the existence of J, K, and the probability of a job entering node K upon completion of service at node J. All jobs enter the network from the origin and exit the network through the destination.

A state of the system is defined by the number of jobs n_i in each node i. If a queue i is serviced by m devices, then for $n_i \leq m$ all jobs are being serviced, while for $n_i > m$ there are m jobs being serviced and $m-n_i$ jobs waiting for service. The sum of all n_i's must equal the constant, N, the total number of jobs in the system

at all times. Thus a state is defined by a state vector $(n_1,n_2,\ldots,n_L)$ where L is the number of nodes of the system and n_i is the number of jobs in node i. The template for the state vectors of a system is the list of queue labels defining the state vector convention. Thus for the network of figure 2, if (Q1,Q2,Q3,Q4,Q5) is the template then the system is in state (2,1,1,0,0). If (Q3,Q1,Q5,Q2, Q4) is the template then the state vector is (1, 2,0,1,0).

Branching probabilities are input as polynomials. The service rate for a device is defined by a mean service time and is also considered a polynomial. The analytic technique of ASQ and the polynomial representation of all parameters allows information of the system to be a function of arbitrarily many variables.

A polynomial may consist of any number of variables (including none). Exponents in the polynomial must be integers. Thus a branching probability, mean service time, steady state probability, etc., could be:

$$x$$
$$1-x$$
$$4x^3 + x^2y - 0.8x + yz - z^{-3} - 0.7$$
$$0.427$$

with input/output formats:

```
X
1-X
4*X↑3+X↑2*Y-0.8*X-Z↑-3-0.7
0.427
```

No provision is made for inputting fractions, as the polynomial manipulation routines will not do division. If necessary, a "fraction" is output by the program by printing a numerator and denominator. However, this does not correspond to any internal polynomial representation in the program.

A network is described to the program in the following manner: (Program responses are in upper case, user responses in dropped lower case)

```
*INPUT EDGES:  NODE/PROBABILITY/NODE;
```

(The user must input all edges of the network. This establishes all nodes and their labels, all branches and branching probabilities, and all joins. Consider the network in figure 2 in this and the following example. The user would respond:)

```
q1/0.43/q3;
q1/0.57/q4;
q2/2*m↑3/q3;
q2/v/q4;
q2/1-2*m↑3-v/q5;
→;
```

The character ";" is the line delimiter and "→" signals the end of user input.

```
* INPUT BRANCHES FROM ORIGIN:  PROBABILITY/NODE/;
x/q1;
1-x/q2;
→;
* INPUT BRANCHES TO DESTINATION: NODE/PROBABILITY/;
q3/1/;
q4/1/;
q5/1/;
→;
```

(The number of devices in a node is implied by the number of different mean service times of the node. The node label must be input and the mean service time for 1 job in the node, the mean service time for 2 jobs in the node, etc. In the case of more jobs than devices in the specified node the program uses the mean service time for all devices occupied, which is the last time input on the list. For example, the devices of Q2 each have a mean service time of $w-w^2$. Thus the mean service time for 2 or more jobs in node Q2 is $\frac{1}{2}(w-w^2)$.)

```
* INPUT MEAN SERVICE TIME(S):  NODE/TIME(1 JOB)/
     TIME(2 JOBS)/.../;
q1/0.183/;
q2/w-w↑2/0.5*w-0.5*w↑2/;
q3/w/;
q4/z+5*w/;
q5/1.4*w/;
→;
* INPUT TOTAL NO. JOBS IN NETWORK :  INTEGER:
4;
```

After the network is thus described the program checks it for any nodes not reachable from the source and for any nodes not reachable to the sink. If no such nodes are found the program accepts a command from the user. If such nodes are found they are output in error message(s). For example, the result of the network of figure 3 being input would be:

```
*NODES NOT CONNECTED FROM SOURCE - (B3 B6 B7)
*NODES NOT CONNECTED FROM SINK - (B5 B6 B7)
```

In such a case the program goes into a mode such that only commands by the user changing the network are accepted. In the case of figure 3, the user could make an acceptable network by adding a join from B5 to the sink, adding a branch from B2 to B3, and deleting B6 and B7.

After an acceptable network is defined to the program, ASQ asks if the user wishes to specify the template.

*DO YOU WISH TO SPECIFY THE TEMPLATE?: YES/NO;
yes;

A yes response requires the user inputting the template vector.

T = (q5,q3,q1,q2,q7,q4,q6);

If the user does not wish to specify the template the program generates it and outputs it by the same format.

The following commands are executable at this stage, with the given output in polynomial form.

sspb/$(n_1,n_2,\ldots,n_L)/(n_1',n_2',\ldots,n_L')/\ldots/$;
outputs an unnormalized steady state probability for each of the specified vectors.

snrm/;
outputs the normalization factor for the steady state probabilities.

tput/;
outputs the throughput of the system.

util/$node_1$/$node_2$/$node_3$/.../;
returns the utilization of each node specified.

assp/;
returns a table of all the unnormalized steady state probabilities.

tble/parameter/;
is a special command, given to generate the parameter desired in polynomial form, output it, and then accept numerical values for substitution into all variables except one - thus making the polynomial a function of one variable. A lower limit, upper limit, and incremental step are then input for this variable - resulting in a table of the parameter values versus the variable values. The maximum and minimum values of the parameter and the corresponding variable values are also output. Parameters may be specified as sspb(n_1, n_2, ..., n_L), snrm, tput, or util(node). For example, suppose the unnormalized steady-state probability of a state (1, 0, 1, 2, 0, 1) is $x^2 + 3xy - 4y^2 + 0.38z - 0.4$ and the user wishes to find the behavior of this parameter as a function of x as x ranges from 2.0 to 2.5 in steps of 0.1 while y is set to 0.2 and z to 0.1. The user would enter the following command:

tble/sspb(1, 0, 1, 2, 0, 1)/; resulting in:
*X↑2 + 3*y - 4*Y↑2 + 0.38*Z - 0.4 with the user specifying:

*VAR = x;
* Y = 0.2;
* Z = 0.1;
*LOWLIM/HIGHLIM/STEP/ = 2/2.5/0.1/;

results in the following table

2.000000000	4.678000000
2.100000000	5.148000000
2.200000000	5.638000000
2.300000000	6.148000000
2.400000000	6.678000000
2.500000000	7.178000000
*MAX 2.500000000	7.178000000
*MIN 2.000000000	4.678000000

end/;
terminates the program.

This completes the discussion of user options with the program.

Some Applications

1. Optimization of branching

Consider Fig. 2. Suppose queues 1 and 2 handle the same sorts of requests. Thus we might vary x, the fraction of jobs which go to queue 1. Suppose we wish to find the value of x which maximises throughput for this network, with v = 0.2, m = 0.3, z =0, w = 0.1. We compute the throughput as a polynomial in x, and then using the "tble" function, proceed to get a table of throughput for different values of x. The maximum throughput in the table and the value of x for which it is achieved will be output automatically.

2. Optimum service rates

Consider Fig. 1. Suppose all the parameters except the service rates are fixed and given. Our objective is to buy devices for queue 1 and queue 2. Various kinds of devices are available, and

generally the more expensive devices have faster service. Given a budget, our objective is to share the budget between the two devices to maximise throughput. Let C be the total budget, and C_1 the amount spent on device 1. We input the mean service time for queues 1 and 2 as a function of C and C_1. We may then obtain the throughput as a polynomial in C and C_1. The function of throughput will be extremely useful to management in deciding how much it wants to spend on each device.

3. Evaluating several alternative devices.

Suppose for a given network, we wish to determine the throughput for several alternative values of the parameters. We may obtain the throughput as a polynomial in all the parameters that have to be varied. Once the polynomial is obtained, it is merely evaluated for each set of values of the parameters.

Conclusions

The automated analysis of queueing models is undoubtedly helpful to the computer systems analyst. Irani and Wallace [5] have discussed a system for the conversational design of queueing networks. However our system is different in one very important respect: we are primarily interested in obtaining algebraic solutions. Algebraic and numerical solutions together can be very helpful to the analyst, and in system optimization.

We have discussed the ASQ system which will provide algebraic solutions for queueing network models. We have discussed distribution independent service disciplines and shown how the results of the ASQ system are valid for networks with these service disciplines and non-exponential service distributions with rational Laplace transforms. A partially observable Markov process model was developed to describe the sequence of CPU burst times generated by a given job. The assumption commonly made in queueing models that successive service times are independent is not valid, and the partially observable Markov process is a more realistic description of the situation. It was shown that the results of the ASQ system were valid, even if successive service times were not independent in queues with distribution independent service disciplines. Queueing networks with different types of jobs were discussed. Job typing will be implemented in ASQ in the future. The importance of local balance and the ease with which apparently complex queueing networks can be analyzed using local balance was demonstrated. It may be noted that ASQ will handle most of the queues analysed by Baskett and Palacios [1], Baskett [2], Buzen [7, 8], Gordon and Newell [9], Jackson [6], and Chandy [3].

ASQ is written in LISP 1.5.9 as implemented at the University of Texas at Austin. The program can be run either by batch or interactively on the CDC 6600/6400 system with only minor differences in the commands and command sequence. Since the program is written in a straightforward manner, a user with a basic knowledge of LISP and the structure of the program can tailor it to suit his own needs. ASQ was written in LISP because of the availability of polynomial routines written in that language. The authors would like to acknowledge the invaluable assistance of Don Towsley who prepared the routines.

References

1. F. Baskett and F. P. Gomez. Processor sharing in a central server queueing model of multiprogramming with applications, Proc. Of the Sixth Annual Princeton Conference on Information Sciences and Systems, Princeton University, Princeton, N.J. (March 1972).

2. F. Baskett. Mathematical models of multiprogrammed computer systems, TSN-17, Computation Center Report, The University of Texas at Austin (January 1971).

3. K. M. Chandy. The analysis and solutions for general queueing networks, Proc. of the Sixth Annual Princeton Conference on Information Sciences and Systems, Princeton University, Princeton, N.J. (March 1972).

4. P. J. Denning and S. C. Schwartz. Properties of the orking- et Model. Comm ACM, vol. 15, No.3, (March 1972), pp.191-198.

5. K. B. Irani and V. L. Wallace. On network linguistics and the conversational design of queueing networks. J. ACM, Vol 18, No. 4, (October 1971), pp. 616-629.

6. J. R. Jackson. Jobshop-like queueing systems Man. Sci. Vol. 10, (1963), pp. 131-142.

7. J. Buzen. Analysis of system bottlenecks using a queueing network model. Proc. ACM-SIGOPS Workshop on System Performance Evaluation, Cambridge, Mass. (April 1971). pp. 82-103.

References cont'd

8. J. Buzen, Queueing network models of multi-programming. Ph.D. Dissertation, Division of Engineering and Applied Physics, Harvard University, Cambridge, Mass. (June 1971).

9. W. J. Gordon and G. P. Newell. Closed queueing systems with exponential servers. Operations Research, Vol. 15, No. 2, (April 1967), pp.254-265.

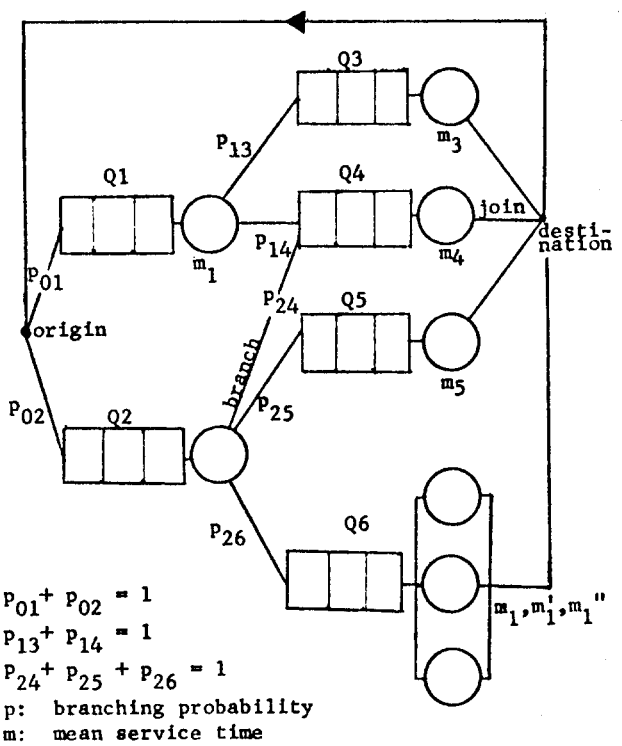


Figure 1

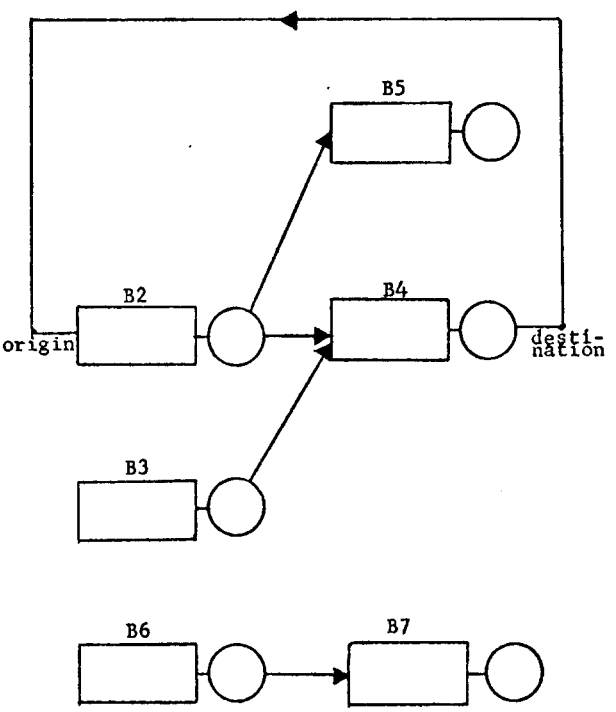


Figure 3

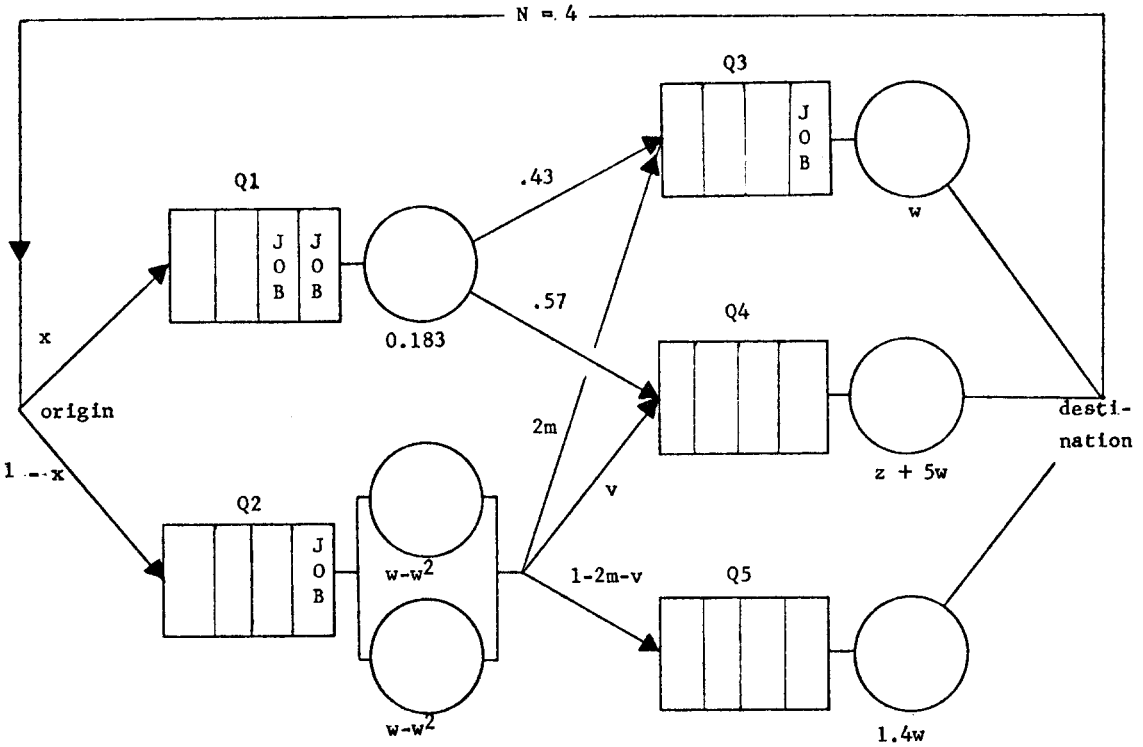


Figure 2

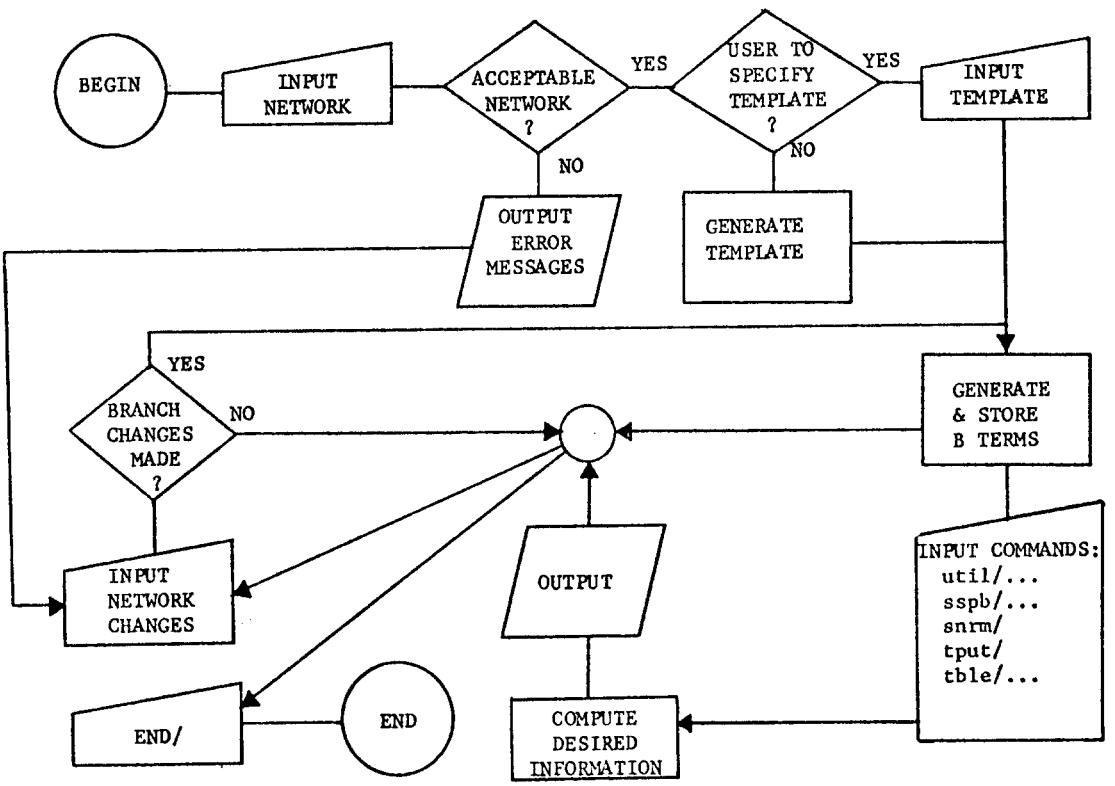
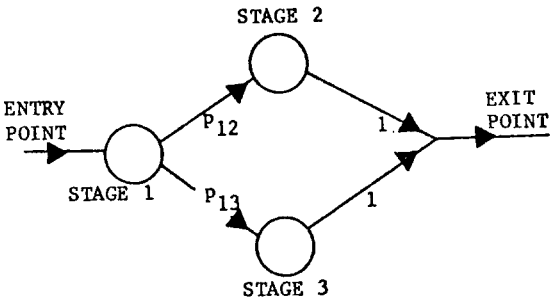
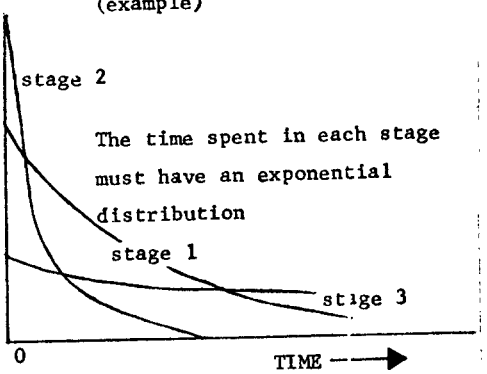


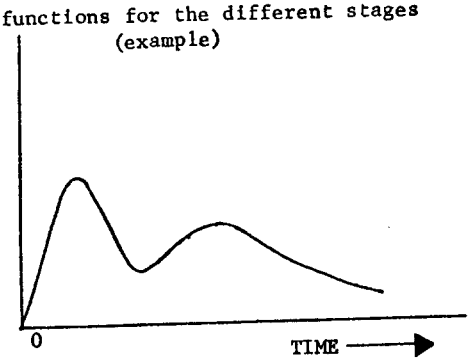
Figure 4



The stages of an EN distribution (example)



Density functions for the different stages (example)



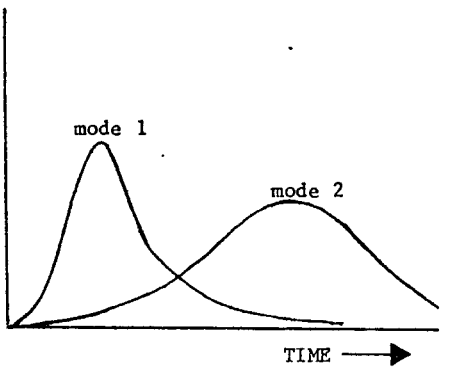
Density function for the EN distribution

Figure 5

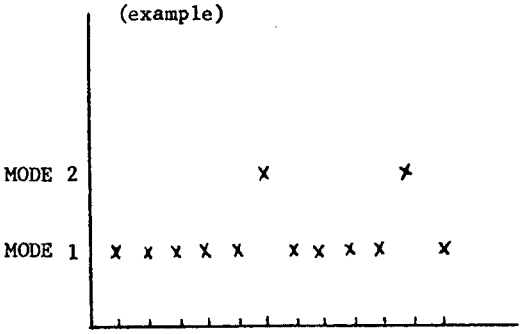
MODE TRANSITION MATRIX:

		to MODE 1	MODE 2
from	MODE 1	0.9	0.1
	MODE 2	0.8	0.2

(example of a mode transition matrix)



CPU burst time distributions in each mode (example)



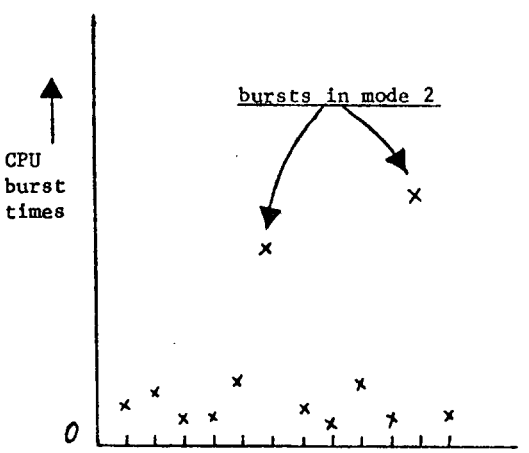


Figure 6