1980

Testing Network-of-Queues Software

Herb Schwetman

Report Number:
80-330

Testing Network-Of-Queues Software

Herb Schwetman

Purdue University
Department of Computer Science
West Lafayette, Ind. 47907

CSD-TR 330

Testing Network-of-Queues Software

Herb Schwetman*
Department of Computer Sciences
The University of Helsinki
Helsinki, Finland

ABSTRACT

With the advent of new techniques for solving networks-of-queues models, there is an emerging need for procedures which facilitate the testing of the programs which implement these techniques. Often, the complexity of the models prohibits traditional "solutions by hand" of any test cases. Therefore, the implementer is forced to devise alternative test procedures.

This paper presents a set of procedures and test problems which have been used to eventually demonstrate correct operation of a solution program. This set, if utilized, will test all features of the complete family of models with "product-form" solutions.

*The author's permanent address is Department of Computer Sciences, Purdue University, West Lafayette, Indiana 47907

NETWORK OF QUEUES SOFTWARE

The past several years have seen the development of networks-of-queues models as tools which can aid in the analysis of computer system performance. All of these are based on the product-form solution, which was expanded and summarized in [BCMP75]. One of the most widely used versions of this family of models is the central server model of Buzen [Buze71]. Since that development, exact solution techniques have been devised so that models embodying all features of the family of models with product form solution can be efficiently solved. The paper by Kienzle and Sevcik [KiSe79] ennumerates many of these techniques and the programs which impelement them.

One of the most general programs [BBS77] uses the convolution technique [ReKo75]. Since that time, Reiser and Lavenberg [ReLa78] have developed the mean-value technique, which is much simpler and less susceptible to numerical instabilities than its predecessors. A paper by Sauer and MacNair [SaMc79] summarizes a set of design goals for networks-of-queues software.

When implementing a program to produce solutions, there are at least four ways of obtaining answers which can be used for comparisons with the solutions from the program; these are (1) use of an equivalent discrete-event simulation model, (2) use of models which can be directly solved using an alternative technique, (3) use of models which have been previously solved, and (4) use of combinations of models to test consistency of results. Each of these verification methods has its drawbacks. For example, simulation models rarely produce exact results. Models which can be solved using alternative techniques usually do not possess all of the features implemented in the new program. Use of previously solved models is difficult because of the lack of published works which contain all of the information required to make comparisons.

In the remainder of this paper, we present a detailed set of examples and test cases which can be used to verify the output of a solution program. Simulation is not covered, since this is an obvious technique and has been discussed elsewhere [Saue79]. While accurate solutions cannot be guaranteed, experience has shown that the chances for accurate results will be enhanced, if the examples are correctly executed. Where relevant, references are provided to published works which are complete enough to be of use in the present context.

The notation and terminology used in this paper are summarized in Table 1.

K	the number of devices in a closed model
N	the number of customers or jobs in a model
Si	mean service interval at device i
Ui	utilization of device i
ni	mean number of customers at device i
Xi	throughput rate of device i
Ri	mean response time of device i

When job classes are used, the class designator is placed in parentheses.

Queue service discipline are denoted as follows:

- FCFS first come, first served,
- IS infinite number of servers
- PS processor sharing
- LCFS last come, first served, preempt.

Table 1
Notation and Terminology

MODELS WITH ALTERNATE SOLUTIONS

The two queue, cyclic network, shown in Figure 1, is the simplest closed network which can be imagined.

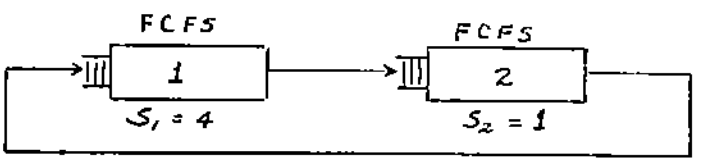


Figure 1
Tandem, Cyclic Network

$p = S1/S2$	$P0 = 1/(1 + p + p^2 + \ldots + p^N)$
$U1 = 1 - P0$	$U2 = 1 - p^N * P0$
$n1 = \sum_i i * p^i * P0$	$n2 = N - n1$
$X1 = U1/S1$	$X2 = X1$
$R1 = n1/X1$	$R2 = n2/X2$

Table 2
Solutions for Tandem, Cyclic Network

Assuming that the service times at each queue are given as negative exponential probability distributions with means of S1 and S2 respectively, this network can be solved using techniques from basic queueing theory [Klei75]. These solutions are shown in Table 2. For the network shown in Figure 1, some representative solutions are given in Table 3.

N	U1	n1	X1	R1	U2	n2	X2	R2
1	.800	.800	.200	.400	.200	.200	.200	1.00
2	.952	1.71	.238	7.20	.238	.286	.238	1.20
3	.988	2.68	.247	10.9	.247	.318	.247	1.29

Table 3
Solutions for Figure 1

It can be noticed that several "common sense" criteria can be applied to solutions. For example, when N is 1, then the response time is equal to the service (i.e. Ri = Si), and the mean queue length is equal to the utilization (ni = Ui). Other checks can be made such as insuring that the mean queue lengths sum to N ($\Sigma ni = N$) and that the throughput rates balance (i.e. that the output rate for a queue is the sum of the input rates, etc.). In the example, this last criterion means that X1 = X2 for all N.

Another model which can be solved is the machine repairman model, first used to model computer systems by Scherr [Sche67]. In networks-of-queues notation, such a model can constructed in several ways. The simplest representation is given in Figure 2, but other representations will be studied in a later section.

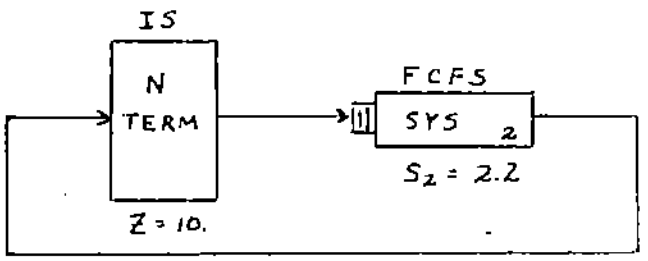


Figure 2
Network of Queues Representation of Machine Repairman Model

The model in Figure 2 represents a system as a single-server, FCFS queue and a collection of terminals as a

queue with an infinite number of servers (i.e. no transaction ever waits to obtain a terminal). The parameters of the model are the mean think time Z (the service time at a terminal), the mean service time at the system S2, and the number of active terminals N. The solution for this model appears in many places, including [HiLi67,p. 305]. The relevant formulae are shown in Table 4.

$$P0 = \left[\sum_i \frac{N!}{(N-i)!} \left(\frac{S2}{Z}\right)^i \right]^{-1}$$

$$U2 = 1 - P0$$

$$n2 = N - Z*U2/S2$$

$$R2 = N*S2/U2 - Z$$

Table 4
Solution Formulae for Machine Repairman Model

For the model shown in Figure 2, representative answers are listed in Table 5.

N	U2	n2	X2	R2
1	.180	.180	.082	2.20
2	.349	.412	.159	2.60
3	.504	.711	.229	3.11
4	.639	1.09	.291	3.77
5	.753	1.58	.342	4.61

Table 5
Solutions for Machine Repairman Model

These two models, the cyclic, tandem-queues model and the machine-repairman model, represent the kinds of models which are readily solvable by direct methods. These are very useful when entering the initial phases of testing a program.

USE OF EQUIVALENT MODELS

The next kind of testing uses different versions of the same model which should produce identical solutions. One example of equivalent versions is the use of two job classes, each with the same attributes. Use of this type of model can help verify that a solution program for models with multiple job classes is producing correct results. The model shown in Figure 3 is taken from Buzen's article [Buze73].

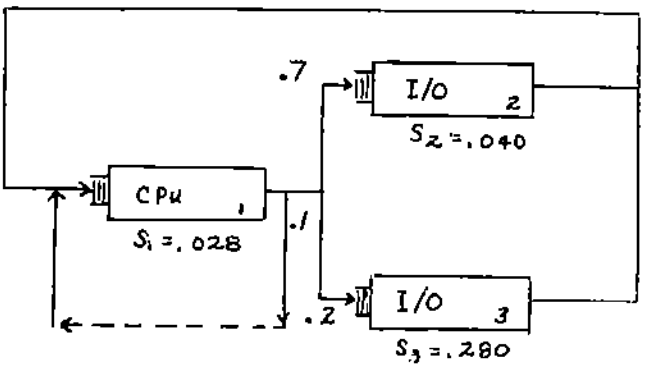


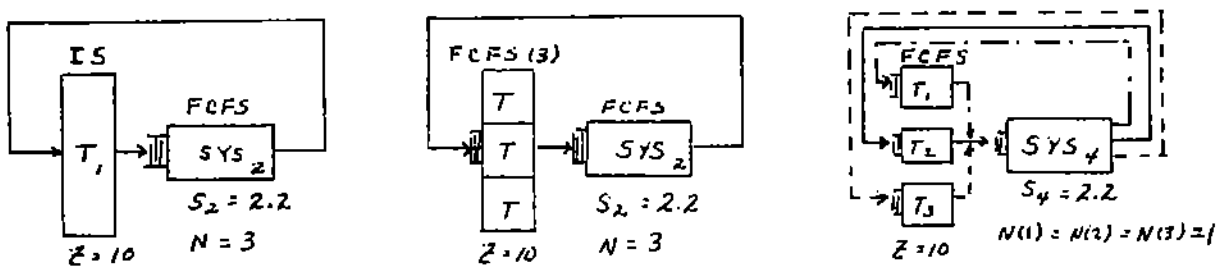
Figure 3
Central Server Model

In this test, two job classes are defined, with the same service times and branching probabilities for each class. Then, the model is executed with various combinations of jobs from these two classes, but so that the total number of jobs remains constant (e.g. N(1) = 1, N(2) = 3, and then N(1) = 4, N(2) = 0). The global solutions for these different versions should be the same. The solution for this model with four jobs active is given in Table 6.

device	Ui	ni	Xi	Ri
1 CPU	.456	.737	16.2	.045
2 I/O	.456	.737	11.4	.065
3 I/O	.912	2.53	3.26	.775

Table 6
Solution for Central Server Model
(N = 4)

Another form of equivalent models can be used to test models with additional features. For example, the models shown in Figures 4a, 4b, and 4c should all yield identical results.



(a) (b) (c)

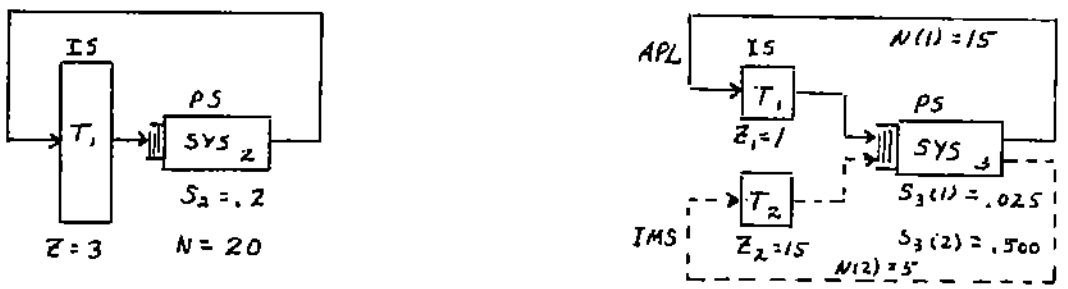
Figure 4
Three Equivalent Models

The model in Figure 4a was previously seen in Figure 2. The model in Figure 4b tests the multi-server-queue feature of a solution program, and the model in Figure 4c tests the use of job classes with different branching probabilities. The correct solution for these models is given in Table 4, in the row for three jobs active (N = 3).

USE OF PREVIOUSLY SOLVED MODELS

The "previously solved models" in the title of this section refers to both published solutions and solutions obtained from an existing program which is known to produce correct results. However, in this section, we will discuss only solutions which have appeared in journal articles.

The first two models (see Figures 5a and 5b) appeared in an article by Reiser [Reis76].



(a) (b)

Figure 5
Two Models by Reiser

The solutions for these are given in Tables 6 and 7 respectively.

device	Ui	ni	Xi	Ri
1 TERM	.716*	14.3	4.77	3.00
2 SYS	.954	5.68	4.77	1.19

*per terminal

Table 6
Solution for Model in Figure 5a

device	class	Ui	ni	Xi	Ri
1 TERM	APL	.955*	14.3	14.3	1.00
2 TERM	IMS	.941*	4.71	.314	15.0
3 SYS	APL	.358	.677	14.3	.047
	IMS	.157	.293	.314	.934
	TOTAL	.515	.970	14.6	.066

*per terminal

Table 7
Solution for Model in Figure 5b

The next model is from an article by Sauer [Saue79]. It is of interest because it does not have the topology of a central server model. Also, this model (as well as the model of Figure 5b) has a queue, the CPU, where jobs in different classes have different service intervals. This model is shown in Figure 6 and the solutions are in Table 8.

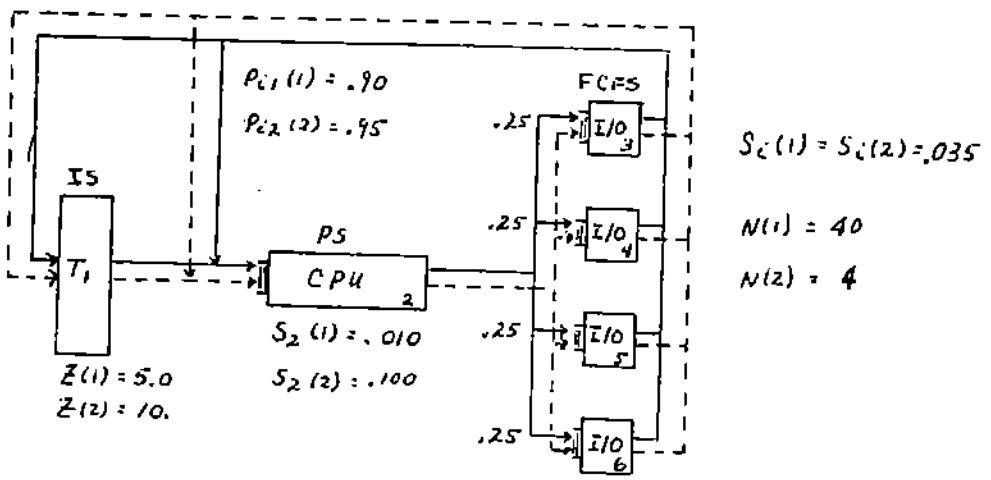


Figure 6
Model by Sauer

device	class	Ui	ni	Xi	Ri
1 TERM	1	.771*	30.8	6.16	5.00
	2	.428*	1.71	.171	10.0
	BOTH		32.5	6.34	
2 CPU	1	.616	4.39	61.6	.071
	2	.342	2.02	3.42	.589
	BOTH	.959	6.41	65.1	.098
3-6 I/O	1	.539	1.20	15.4	.078
	2	.030	.069	.855	.080
	BOTH	.569	1.27	16.3	.078

*per terminal

Table 8
Solution for Model in Figure 6

TESTING OTHER FEATURES

There are three features of models with product-form solutions which have not been covered so far: multi-server queues with class dependent service times, class switching by jobs, and general service functions. None of these were included in the program under test but are discussed here, in order to complete the set of examples. The model in Figure 7 has a queue, the CPU, which has two servers and jobs in different classes have different service requirements.

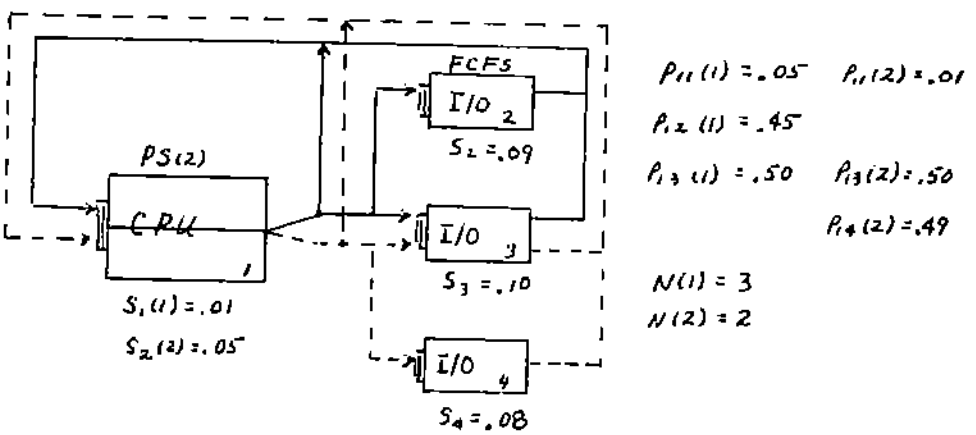


Figure 7
Model with a Multi-server Queue
and Class Dependent Service

The solution for N(1) = 3, N(2) = 2 was obtained from the program in [BBS77] and appears in Table 9.

device	class	Ui	ni	Xi	Ri
1 CPU(2)	1	.095	.127	12.1	.011
	2	.307	.374	7.32	.051
	BOTH	.402	.501	19.4	.026
2 I/O	1	.488	.749	5.42	.138
	2	0	0	0	0
	BOTH	.488	.749	5.42	.138
3 I/O	1	.603	2.12	6.03	.353
	2	.366	1.29	3.66	.352
	BOTH	.969	3.41	9.69	.352
4 I/O	1	0	0	0	0
	2	.287	.337	3.59	.094
	BOTH	.287	.337	3.59	.094

Table 9
Solution for Model in Figure 7

A model which illustrates class switching appears, in complete form, as an example in [BBS77]. The use of general service functions (other than to model multi-server queues) has been used in only a limited way, and no example of this has been included here. Normally, if the multi-server features function correctly, then the generalized service-function features ought to work, as they both are usually implemented using the same code.

CLOSING REMARKS

The need for an article such as this became apparent when the author undertook the implementation of a new solution program, and there was no easy access to an existing program which contained all of the features of the new program. Therefore, validation of the new program had to be accomplished by using the techniques presented here.

The examples which appear here were chosen with care. Many of them pointed out bugs or errors in the new program being developed. For example, Reiser's models (Figures 5a and 5b) may tax the arithmetic precision of a given computer. Sauer's example (Figure 6), since it does not have the configuration of the common central server model, uncovered errors in solving for the relative arrival rates. The multi-server and multi-class models of Figures 3 and 4 were essential to debugging the relevant portions of the new program.

One point emerged from the testing described above, namely that a correct program will produce results which agree very closely with "correct" results (e.g. to within the third decimal place). If the results are not in close agreement, the

new program is probably in error, and the lack of agreement cannot be attributed to other sources, such as the differences in word lengths or arithmetic properties of two computers.

It should be pointed out that this collection of examples is probably not complete, in the sense of exercising every path and every feature in a solution program. Just because a program can correctly solve these examples does not guarantee the absence of errors. However, these examples can aid in the implementation of a solution program and should speed up the development process.

It is this author's belief that there would be great benefit if a collection of results of complete models were available, where complete means with enough detail so that the models can be solved and the answers compared. Of course, the accuracy of these "standard solutions" must be verified somehow. The examples in this paper could be the start of such a collection, but other examples should be added, e.g. example of models with larger numbers of devices, classes and customers. The existence of such a collection could help many researchers in the area of models of computer systems.

ACKNOWLEDGEMENT

This work was done while the author was a Fullbright/Hayes Lecturer in the Department of Computer Sciences of the University of Helsinki. Some of the examples were run by Steve Tolopka, at Purdue University.

LIST OF REFERENCES

BBS77 Balbo, S., Bruell, S. and H. Schwetman, "Customer classes and closed networks models - a solution technique", Proc. IFIP Congress 77, North-Holland, 1977, p. 559.

BCMP75 Baskett, F., Chandy, M., Muntz, R. and J. Palacios, "Open, closed and mixed networks with different classes of customers", Jour. ACM (22,2), April, 1975, p. 248.

Buze71 Buzen, J., "Analysis of system bottlenecks using a queueing network model", Proc. ACM/SIGOPS Workshop on System Performance Evaluation, 1971, p. 82.

Buze73 Buzen, J., "Computational algorithms for closed queueing networks with exponential servers", Comm. ACM (16,9), Sept. 1973, p. 527.

HiLi67 Hillier, F. and G. Lieberman, Introduction to Operations Research, Holden-Day, 1967.

KiSe79 Kienzle, M. and K. Sevcik, "Survey of analytic queueing network models of computer system", Proc. The Conference on Simulation, Measurement, and Modeling of Computer Systems (ACM/SIGMETRICS), Aug. 1979, p. 113.

Klei75 Kleinrock, L., Queueing Systems I, John Wiley, 1975.

Reis76 Reiser, M., "Interactive modeling of computer systems", IBM Sys. Jour. (15,4), 1976, p. 309.

ReKo75 Reiser, M. and H. Kobayashi, "Queueing networks with multiple closed chains: theory and computational algorithms", IBM Jour. Res. Dev., May 1975, p. 283.

ReLa78 Reiser, M. and S. Lavenberg, "Mean value analysis of closed, multichain queueing networks", IBM T.J. Watson Research Center, RC 7023.

SaMa79 Sauer, C. and E. MacNair, "Queueing network software for systems modelling", Software Prac. Exper., (9), 1979, p. 369.

Saue79 Sauer, C., "Confidence intervals for queueing simulations of comuter systems", Perf. Eval. Rev. (ACM/SIGMETRICS), (8,1-2), Summer, 1979, p. 45.

Sche67 Scherr, A., An Analysis of a Time-Shared System, MIT Press, 1967.