# THE RESEARCH QUEUEING PACKAGE

## Past, Present and Future

Charles H. Sauer, Edward A. MacNair, James F. Kurose

IBM Thomas J. Watson Research Center

Yorktown Heights, New York 10598

(914) 945-1542

**Abstract:** Queueing networks are important as performance models of systems where performance is principally affected by contention for resources. Such systems include computer systems, communication networks, office systems and manufacturing lines. In order to effectively use queueing networks as performance models, appropriate software is necessary for definition of the networks to be solved, for solution of the networks (by numerical, approximate and/or simulation methods) and for examination of the performance measures obtained. One of the most widely known and influential pieces of queueing network software is the Research Queueing Package (RESQ). This paper discusses the evolution of RESQ and plans for further RESQ development. (Note – RESQ is not available outside of IBM except under a few special agreements.)

Key Words and Phrases: performance modeling, queueing networks, simulation, modeling software

## INTRODUCTION

Many physical systems, including computing systems, communication networks, office systems and manufacturing lines, are heavily dependent on sharing of resources. Sharing of resources necessarily leads to contention, i.e., queueing, for resources. Contention and queueing for resources are typically quite difficult to quantify when estimating system performance.

Queueing models have been used for decades in studying the performance of manufacturing lines, communication networks and similar systems. In the last two decades queueing models have become important as performance models of computing systems. Since office systems have become heavily dependent on computing and communication, queueing models are appropriate in office system performance evaluation. These models are often networks of queues because of the interactions of system resources. For a general discussion of queueing network models, see Sauer and Chandy[1] and recent special issues of Computing Surveys (September 1978) and Computer (April 1980).

For queueing network models to be used effectively, appropriate software is necessary to construct models and to obtain solutions for models. One of the most widely known and influential pieces of queueing network software is the Research Queueing Package (RESQ)[2-5]. (RESQ is restricted to IBM internal use except for special agreements with a few universities.) Other pieces of software influenced by RESQ include the

Queueing Network Analysis Package (QNAP)[6] and the Performance Analyst's Workbench System (PAWS)[7].

RESQ is important and influential because of (1) the "extended" queueing networks associated with RESQ, (2) the diagram language used to informally represent queueing networks to be handled by RESQ, (3) the user language and machine interfaces used to formally represent queueing networks and their solutions and (4) the multiple solution methods of RESQ, including the research effort that has gone into their design and implementation. This paper discusses these points from a historical viewpoint and discusses the expected future evolution of RESQ.

## QUEUEING NETWORK MODELS

The following discussion will primarily use computing system terminology and assume the reader can provide the analogous terminology for other systems. A typical queueing network model consists of a set of queues (corresponding to resources in a computer system) and a set of jobs (which correspond to processes in a computer system, users at terminals, messages sent from computer to computer, etc., depending on the system). The individual queues are usually described in terms of types of resources, numbers of units of resources, queueing (scheduling) disciplines and probability distributions for the service times of jobs at the queues. The jobs are described by their individual characteristics, by their routing from queue to queue (corresponding to the sequence of resource requirements in the system) and by their arrival processes (and

departure procedures).

Much of the research on queueing network models has focused on methods for obtaining solutions, i.e., performance estimates, for the models. Efficient numerical algorithms have been developed for networks with a product form solution[1,8-12]. However, there are many system characteristics which preclude a product form solution, e.g., priority scheduling or simultaneous resource possession. For models with these characteristics and more than a few queues and/or jobs the only solution methods available are approximate numerical methods[1,13,14] and simulation. Specialized simulation techniques have been developed which apply to simulation of queueing networks[1,15].

RESQ incorporates both numerical and simulation solution methods. Though RESQ includes simulation components, we do not consider RESQ to be a simulation language. Rather, we consider RESQ to be a modeling language. We make the distinction primarily because of the higher level of abstraction of RESQ elements, as compared to popular simulation languages, and also because of the numerical (non-simulation) solution methods provided in RESQ.

## EXTENDED QUEUEING NETWORKS

In order to facilitate more accurate representation of systems, the queueing networks of RESQ have been designed to include and naturally build upon the category of networks with product form solution. Some of

the elements are obvious generalizations of product form elements, for example queues with general (e.g., priority) scheduling disciplines. Other generalizations of product form networks include (1) capabilities for marking jobs with information (such as message length for a job representing a message in a communication network) and (2) routing rules dependent on the current network state (e.g. queue lengths) as well as the usual probabilistic routing rules.

In addition to allowing the above described characteristics which usually violate product form solution conditions, we provide in RESQ new network elements and refer to the resulting category of networks as "extended" queueing networks[16]. We restrict attention to the most important of these elements, the "passive queue". We refer to traditional queues as "active queues." One of the limitations of a network consisting only of active queues is that a job can only hold one resource at a time. This can be a severe restriction in studying systems in which a job requires several resources simultaneously. For example, a program requires memory as well as a CPU before it can be run, but most traditional queueing models will ignore either memory contention or CPU contention. In extended queueing networks a job can hold resources at several passive queues and one active queue simultaneously.

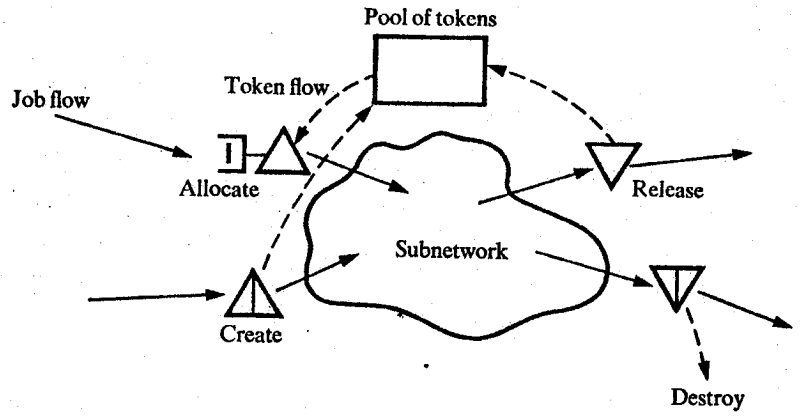


Figure 1. A passive queue.

A passive queue consists of a set of "allocate nodes", a set of "release nodes", a set of "create nodes", a set of "destroy nodes", and a pool of identical "tokens" of a resource. A job joins a passive queue when it arrives at an allocate node. Upon arrival the job requests one or more tokens. If sufficient tokens are available, the requested number of tokens is allocated to the job, which then moves on to another queue of the network without delay. However, the job belongs to the queue from which it received the tokens as long as it holds the tokens. If insufficient tokens are available, the job waits until enough become available and then immediately moves on through the network after receiving them. When several jobs wait for tokens of a passive queue, they are allocated tokens according to a specified scheduling discipline. A job gives up tokens, and thus leaves the corresponding passive queue, when it is routed through a release node of the queue. The job passes through the release node instantaneously. Create nodes have no effect on the job; jobs passing through a create node simply add new tokens to the pool.

Destroy nodes are similar to release nodes but do not return the tokens to the pool. Jobs pass instantaneously through create and destroy nodes. See Figure 1.

The terms "active queue" and "passive queue" are intended to indicate the nature of the queue's effect on a job's use of a server or token, respectively, and of the relative dominance of the modeled resources. With an active queue the length of time a job holds a server is entirely determined by the characteristics of that queue and the jobs at that queue. With a passive queue the length of time a job holds a token is determined entirely by events at other queues.

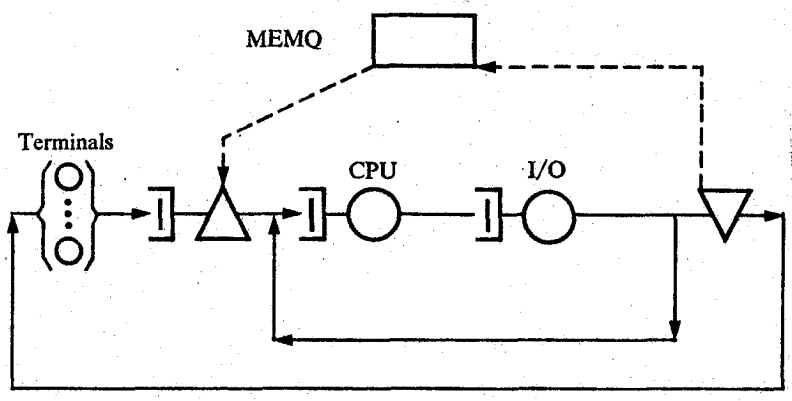


Figure 2. Computer System Model.

Figure 2 shows a simplistic representation of a widely used model of interactive computer systems. The resources represented by active queues are the terminals, CPU and I/O device(s). A passive queue is used to represent memory contention. After a think time at the terminal, a user keys in a command. A job representing the process executing the command

requests memory. After receiving memory the job alternates between CPU and I/O activities until the command is finished. The job then releases its memory and returns to the terminals queue for another thinking and keying time. For other examples of passive queues and extended queueing networks, see[16,17].

## RESQ HISTORY

The original solution components of RESQ, QNET4 (numerical solution) and APLOMB (simulation), were separately developed in 1974 by M. Reiser at the IBM Thomas J. Watson Research Center and C.H. Sauer at the University of Texas, respectively.

QNET4 was initially implemented in APL and subsequently reimplemented in PL/I. Though QNET4 is essentially unchanged, it represents the state of the art of the "convolution" algorithm it uses. (The Research Queueing Package Version 2 (RESQ2) provides an alternate computational algorithm, Mean Value Analysis[10], for the same class of networks handled by QNET4.)

APLOMB was initially implemented in Fortran. Two special features of APLOMB are (1) the use of extended queueing networks (including passive queues) used to represent models and (2) the provision of statistical output analysis techniques (including confidence interval estimation and stopping rules). APLOMB has been (and is being) continually revised and improved over the years. In late 1976 APLOMB was translated from Fortran to PL/I.

In the spring of 1976, QNET4 and APLOMB were provided with a common user interface implemented by E.A. MacNair. The three programs became collectively known as RESQ. This prototype version of RESQ used the APL QNET4 with the interface implemented in APL[2,3]. In the spring of 1977, a new version of RESQ, "RESQ1," was developed. RESQ1 was implemented entirely in PL/I, though some components were duplicated in APL for users who preferred that environment to CMS or TSO.

In the summer of 1978, an entirely new user interface was designed to overcome a number of fundamental limitations of the original interface. This new version is known as "RESQ2." It became operational in July 1980.

## TECHNICAL CONTRIBUTIONS OF RESQ1

Two of the primary technical contributions of RESQ1 are the extended queueing networks and the diagram language for describing extended queueing networks. The extensions, especially passive queues, make queueing networks a powerful framework for abstracting the essential characteristics of systems' performance. The diagram language provides a concise means of describing systems, even when actually constructing a model with RESQ is not contemplated. The extended queueing networks and diagram language have had a strong influence outside of IBM, e.g., have influenced software packages such as QNAP[6] and PAWS[7]. Though RESQ2 is a much more powerful tool than RESQ1, there have been only relatively minor additions needed in the extended queueing networks of RESQ1 in the

development of RESQ2.

Besides the extended queueing networks and diagram language, the focus of RESQ1 development was the simulation portion, APLOMB. The numerical solution portion, QNET4, remained essentially unchanged from its state before RESQ1 (and remains essentially unchanged today). However, since APLOMB provides the simulation capability needed to solve extended queueing networks, APLOMB continued to evolve as the queueing network extensions were developed. APLOMB and the extended queueing networks were also important in that they provided an impressive demonstration that the regenerative method for confidence intervals[15], new in the literature at that time, had practical application far beyond the "toy" applications in the literature. APLOMB also included a sequential stopping rule for determining simulation run length[18].

The greatest failing of RESQ1 was the rigid language used to formally define and describe the queueing models. Little attention was given to this language, though much effort went into the interactive implementation of the language. An implicit assumption in the language design was that models constructed with RESQ1 would be small in terms of numbers of elements and the language could thus be designed for implementation convenience and efficiency, rather than user convenience and efficiency. Because of this rigidity, the interactive interface (and its subsequent "dialogue file" mode) were inconvenient, at best, for the large system models made attractive by the extended queueing networks and APLOMB.

## RESQ2 DESIGN AND DEVELOPMENT

### RESQ2 language design

The objective in the language design was to provide a language similar in appearance to the RESQ1 dialogues[16] but providing the features and flexibility of a modern programming language. (Block structured programming languages, in particular Pascal, were especially influential, though this influence is not obvious in the actual syntax.)

Some of the changes from the RESQ1 language to the RESQ2 language are simple, yet important, improvements which corrected some of the deficiencies of the RESQ1 language. For example, the RESQ1 dialogues require that queues (and other network elements) be numbered sequentially and referenced by these numbers. The RESQ2 language allows symbolic naming of elements. The RESQ1 dialogues generally allow only numeric constants where numeric values are required. The RESQ2 language allows arbitrary numerical expressions in such places. These expressions may include symbols previously defined to have constant values, symbols representing parameters to be defined by the user before solution begins, and symbols representing values which may vary during a simulation. The RESQ1 dialogues require that the number of queues (and numbers of other elements) be specified at the beginning, forcing the user to make a count and stick to it. The RESQ2 language avoids all such requirements.

In addition to these changes, the RESQ2 language provides two kinds of

"templates" (macro-like constructs) which greatly enhance its power. The use of templates makes it possible to describe networks in a much more "structured" manner (in the sense of structured programming) and to sharply reduce the effort required to construct models. One kind of template, the "queue type," provides the ability to create parameterized definitions of queues. Once a queue type has been defined, it can be repeatedly used (invoked) to define specific instances of queues. Queues defined using queue types have default characteristics specified in the queue type definition; other queue characteristics are specified by parameter values given with the queue type invocation.

The other kind of template, the "submodel," allows definition of a parameterized template of an entire subnetwork, which may be used repeatedly in defining a network. Previous work on programming languages provided little guidance on how such subnetworks should be specified and interfaced with the remainder of a network. Following are a submodel definition for part of the network of Figure 2:

```
SUBMODEL:cssm /*Computer System SubModel*/
   NUMERIC PARAMETERS:pageframes
   DISTRIBUTION PARAMETERS:disktime cputime
   CHAIN PARAMETERS:chn
   NUMERIC IDENTIFIERS:cpiocycles
      CPIOCYCLES:8
   QUEUE:diskq
      TYPE:fcfs
      CLASS LIST:disk
         SERVICE TIMES:disktime
   QUEUE:cpuq
      TYPE:ps /*Processor Sharing*/
      CLASS LIST:cpu
         SERVICE TIMES:cputime
   QUEUE:memory
```

```
      TYPE:passive
      TOKENS:pageframes
      DSPL:fcfs
      ALLOCATE NODE LIST:getmemory
         NUMBERS OF TOKENS TO ALLOCATE:discrete(16,.25;32,.5;48,.25)
      RELEASE NODE LIST:freememory
   CHAIN:chn
      TYPE:external
      INPUT:getmemory
      OUTPUT:freememory
      :getmemory->cpu->disk
      :disk->freememory cpu;1/cpiocycles 1-1/cpiocycles

END OF SUBMODEL CSSM
```

and a complete model definition which assumes the submodel has been stored in a library:

```
MODEL:csm /*Computer System Model*/
   METHOD:aplomb
   NUMERIC PARAMETERS:thinktime users pageframes
   NUMERIC IDENTIFIERS:cpiocycles
      CPIOCYCLES:8
   DISTRIBUTION IDENTIFIERS:disktime cputime
      DISKTIME:.019 /*mean of exponential*/
      CPUTIME:standard(.05,5) /*mean and coefficient of variation*/
   QUEUE:terminalsq
      TYPE:is
      CLASS LIST:terminals
         SERVICE TIMES:thinktime
   INCLUDE:cssm /*submodel definition from library*/
   INVOCATION:host
      TYPE:cssm
      PAGEFRAMES:pageframes
      DISKTIME:disktime
      CPUTIME:cputime
      CHN:interactiv
   CHAIN:interactiv
      TYPE:closed
      POPULATION:users
      :terminals->host.input
      :host.output->terminals
   QUEUES FOR QUEUEING TIME DIST:host.memory
      VALUES:1 2 3 4 5 6 7 8
   CONFIDENCE INTERVAL METHOD:regenerative
   REGENERATION STATE DEFINITION-
   CHAIN:interactiv
```

```
      NODE LIST:terminals
      REGEN POP:users
      INIT POP:users
   CONFIDENCE LEVEL:90
   SEQUENTIAL STOPPING RULE:yes
      QUEUES TO BE CHECKED:host.memory
         MEASURES:qt
         ALLOWED WIDTHS:10
   SAMPLING PERIOD GUIDELINES-
      QUEUES FOR DEPARTURE COUNTS:host.memory
         DEPARTURES:1000
   LIMIT - CP SECONDS:100
   TRACE:no

END
```

The above are examples of dialogue files, i.e., files similar to the interactive dialogue. Upper case corresponds to prompts in the interactive version, and lower case corresponds to replies in the interactive version. In the true interactive mode, there would be additional prompts of the same form as shown above. These additional prompts would receive no reply from the user, thus indicating the end of a subsection of dialogue. For example, in interactive mode, the actual dialogue correpsonding to the above file might be

```
MODEL:csm /*Computer System Model*/
   METHOD:aplomb
   NUMERIC PARAMETERS:thinktime users pageframes
   NUMERIC PARAMETERS:  /*Null response*/
   NUMERIC IDENTIFIERS:cpiocycles

   ...
```

RESQ2 translator

Because the RESQ1 "dialogue files" had become the more important mode of usage of RESQ1, and because the severe limitations of the language and translator for the dialogue files were the major motivation for RESQ2, the focus of the language and translator design was the dialogue file. It was clear that a compiler-like program was necessary to support the new language features.

A key design decision was that the compiler-like translator use recursive descent parsing. Recursive descent has two important advantages for our translator over more recent techniques based on parser generators. (1) Recursive descent is much more flexible in terms of error recovery. (2) More importantly, due of the flexibility of recursive descent, it has been possible for the same translator to operate effectively as an interactive prompter. Having the same translator capable of both "batch" and interactive modes has been remarkably useful in model construction because (1) in interactive mode, it is possible to immediately make revisions or corrections to prior dialogue by escaping to an editor to revise a transcript of the dialogue so far (a dialogue file) and to then continue in prompting mode after the (incomplete, edited) dialogue file has been reparsed and (2) revision of an existing model is possible in mixed mode by deleting portions of the existing dialogue file and using interactive mode for specification of revisions or additions. This mixed mode capability provides the "user friendliness" of interactive mode without losing the flexibility and efficiency of "batch"

mode for development of significant models.

RESQ2 expansion processor

The output of the translator is a highly symbolic form, far removed from the interface expected by the solution components. This is necessarily the case because of the provision of parameters which are left undefined until the model is to be solved. These run-time parameters allow a model to be solved parametrically without retranslation. A hierarchical network definition, with invocations of submodels, cannot be translated into a monolithic network definition until these parameters are specified. Thus a major portion of the RESQ2 implementation has been the "expansion processor," which produces a network definition at the solution component interface from the symbolic translator output. (The term "expansion" is consistent with the analogy between submodels and macros.)

RESQ2 solution methods

An implementation of Mean Value Analysis is becoming the dominant numerical solution component of RESQ2. (Mean Value Analysis and the Convolution algorithm of QNET4 both handle the full class of product form networks[12]. Each has advantages over the other.) A major aspect of the evolution of APLOMB has been a gradual redesign of the data structures and rewriting of the code to get away from APLOMB's Fortran heritage. These efforts were critically necessary to obtain the efficiency (both storage

and run time) and flexibility needed for RESQ2. Additional extensions to APLOMB were needed to support RESQ2 language features. These extensions include simulation time expression/symbol evaluation and submodel support. Though submodels are nominally hidden from the solution components, submodels must be considered in simulation error messages, trace output and evaluation of expressions which involve submodels.

Other extensions to APLOMB are relatively independent of the RESQ2 language. A major area of improvement in APLOMB is in its output analysis capabilities. Confidence intervals obtained by the classical method of independent replications[1] have been added as an alternative to the regenerative method for models where the regenerative method is not practical or appropriate. The regenerative method implementation has been made more rigorous in its determination of regeneration states. The sequential stopping rule has been refined and made more flexible.

A major new feature of APLOMB is an interactive simulation capability. It is now convenient to continue a simulation run after examining results, either because one wants to see results at intermediate points in the run or because one is not satisfied with results at the planned run length or stopping condition.

## RESQ2 PLANS

A number of RESQ2 features remain to be implemented. Some of these are parts of the original design, while others have been added to our plans

more recently. The most important of these features is the "substitution" (hierarchical/hybrid solution) form of invocation of submodels. Substitutions have the potential of greatly reducing the computational expense of model solution, especially where simulation is involved. A hierarchical solution facility such as this is the best hope for making practical the simulation of very large systems. Since there has been very little work in this area outside of a few feasibility studies[19,20,21], the substitution design will continue to evolve after we gain experience with it.

Another new feature will be the addition of the spectral method for confidence intervals[22]. The spectral method provides another practical alternative to the regenerative method for situations where the classical method of independent replications is inappropriate.

Finally, some of our most ambitious plans are in terms of graphics capabilities for RESQ. For some time we have had the ability to produce high quality diagrams of extended queueing networks on graphics devices. We have recently added the ability to produce diagrams using the output of the RESQ2 translator as input to the graphics programs. This is of great benefit in documenting and debugging models. We have also begun work on constructing models directly by drawing a diagram with a light pen and graphics display. Eventually, this may be sufficient to dramatically reduce the need for typed input. In order to achieve maximum usability, all of the graphics facilities are intended to be usable on low resolution devices, even though a higher resolution device is needed to obtain

aesthetically pleasing results.

## ACKNOWLEDGEMENT

We are grateful to E. Jaffe, P. Rosenfeld, M. Reiser, S. Salza and S. Tucci for their contributions to RESQ.

## REFERENCES

1. C.H. Sauer and K.M. Chandy, Computer Systems Performance Modeling, Prentice-Hall, Englewood Cliffs, NJ (1981).
2. C.H. Sauer, M. Reiser and E.A. MacNair, "RESQ - A Package for Solution of Generalized Queueing Networks," Proceedings 1977 National Computer Conference.
3. M. Reiser and C.H. Sauer, "Queueing Network Models: Methods of Solution and their Program Implementation," in K.M. Chandy and R.T. Yeh, editors, Current Trends in Programming Methodology, Volume III: Software Modeling and Its Impact on Performance, Prentice-Hall (1978) pp. 115-167.
4. C.H. Sauer and E.A. MacNair, "Queueing Network Software for Systems Modeling," Software-Practice and Experience 9, 5 (May 1979).
5. C.H. Sauer, E.A. MacNair and S. Salza, "A Language for Extended Queueing Networks," IBM J. of Research and Development 24, 6 (November 1980).
6. D. Merle, D. Potier and M. Veran, "A Tool for Computer System

Performance Analysis," Performance of Computer Installations, Ferrari, D. (editor), North-Holland (1978).

7. K.M. Chandy, J. Misra, R. Berry and D. Neuse, "Simulation Tools in Performance Evaluation," CPEUG 81, (Computer Performance Evaluation Users Group), San Antonio, Texas (November 1981).
8. J.R. Jackson, "Jobshop-like Queueing Systems," Management Science 10, pp. 131-142 (1963).
9. J.P. Buzen, Queueing Network Models of Multiprogramming, Ph.D. Thesis, Harvard University, Cambridge, Mass. (1971). Garland Publishing, New York (1980).
10. M. Reiser and S.S. Lavenberg, "Mean Value Analysis of Closed Multichain Queueing Networks," IBM Research Report RC-7023, Yorktown Heights, NY (March 1978). JACM 27, 2 (April 1980) pp. 313-322.
11. K.M. Chandy and C.H. Sauer, "Computational Algorithms for Product Form Queueing Networks," RC-7950, IBM Research, Yorktown Heights, N.Y. (November 1979). CACM 23, 10 (October 1980).
12. C.H. Sauer, "Computational Algorithms for State-Dependent Queueing Networks," IBM Research Report RC-8698 (February 1981).
13. K.M. Chandy and C.H. Sauer, "Approximate Methods for Analysis of Queueing Network Models of Computer Systems," Computing Surveys 10, 3 pp. 263-280 (September 1978).
14. C.H. Sauer and K.M. Chandy, "Approximate Solution of Queueing Models of Computer Systems," RC-7785, IBM Research, Yorktown Heights, N.Y. (July 1979). Computer 13, 4 (April 1980) pp. 25-32.
15. D.L. Iglehart, "The Regenerative Method for Simulation Analysis,"

in K.M. Chandy and R.T. Yeh, editors, Current Trends in Programming Methodology, Volume III: Software Modeling and Its Impact on Performance. Prentice-Hall (1978).

16. C.H. Sauer and E.A. MacNair, "Computer/Communication System Modeling with Extended Queueing Networks," RC-6654, IBM Research, Yorktown Heights, N.Y. (July 1977).
17. C.H. Sauer, "Passive Queue Models of Computer Networks," Computer Networking Symp., Gaithersburg, Maryland (December 1978).
18. S.S. Lavenberg and C.H. Sauer, "Sequential Stopping Rules for the Regenerative Method of Simulation," IBM J. of Research and Development 21, (Nov. 1977) pp. 545-558.
19. C.H. Sauer, L.S. Woo and W. Chang, "Hybrid Analysis/Simulation: Distributed Networks," RC-6341, IBM Research, Yorktown Heights, N.Y. (June 1976).
20. H.D. Schwetman, "Hybrid Simulation Models of Computer Systems," CACM 21, 9 (Sept. 1978) pp. 718-723.
21. W.W. Chiu and W-M. Chow, "A Performance Model of MVS," IBM Systems Journal 17, 4 (1978) pp. 444-462.
22. P. Heidelberger and P.D. Welch, "A Spectral Method for Confidence Interval Generation and Run Length Control in Simulation," IBM Research Report RC-8264, Yorktown Heights, New York (1980). Also, CACM 24, 4 (April 1981).