Master's Thesis, Dept. of Electrical and Computer Engineering, University of Arizona, 1993. Also PMRL Technical Report 93-15, Dept. of Electrical and Computer Engineering, Univ. of Arizona, July 1993.

ANALYTIC SOLUTION OF STOCHASTIC ACTIVITY NETWORKS WITH EXPONENTIAL AND DETERMINISTIC ACTIVITIES

by

Bhavan Pukhraj Shah

A Thesis Submitted to the Faculty of the

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

In Partial Fulfillment of the Requirements
For the Degree of

MASTER OF SCIENCE
WITH A MAJOR IN ELECTRICAL ENGINEERING

In the Graduate College

THE UNIVERSITY OF ARIZONA

1 9 9 3

STATEMENT BY AUTHOR

This dissertation has been submitted in partial fulfillment of requirements for an advanced degree at The University of Arizona and is deposited in the University Library to be made available to borrowers under rules of the Library.

Brief quotations from this dissertation are allowable without special permission, provided that accurate acknowledgment of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the head of the major department or the Dean of the Graduate College when in his or her judgment the proposed use of the material is in the interests of scholarship. In all other instances, however, permission must be obtained from the author.

SIGNED:_____

APPROVAL BY THESIS DIRECTOR

This thesis has been approved on the date shown below:

_____ _____

William H. Sanders
Assistant Professor of
Electrical and Computer Engineering

Date

To my parents, and wife,
for loving, caring, and guiding me.

ACKNOWLEDGMENTS

There are numerous people I would like to thank for their help in this endeavor. First of all, I would like to thank my graduate committee members Dr. Pamela Nielsen and Dr. Max Liu for reviewing my thesis, and my advisor, Dr. William Sanders for his guidance, inspiration, and support.

I am thankful to Steve West and his group at ADSTAR, IBM Corporation for funding me throughout my Master's program.

I would like to thank all of my colleagues in PMRL. I would like to thank Latha Kant and Bruce McLeod for reading early versions of this thesis. I would like to thank Luai Malhis for being my buddy on IBM project and making an enjoyable environment in PMRL. I would also like to thank Doug Obal for helping me to understand *UltraSAN* and giving me a company during the night hours in the Lab.

Finally, I would like to thank my family, friends and colleagues for their moral support and encouragement.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Stochastic activity networks, which contain only exponentially distributed timed activities, behave as a Markov process, and can hence be solved using known techniques. When deterministically distributed activities are considered together with exponentially distributed activities, solution becomes difficult due to the non-memoryless property of deterministically distributed activity time. Such systems can be solved for their steady state behavior by employing a transient analysis during the activity time of deterministic activities and applying a correction based upon the fraction of time spent in different states during the activity times of deterministic activities. In this thesis, a method to solve stochastic activity networks with deterministic activities concurrently and/or competitively enabled with exponential activities is implemented as a part of a larger software package known as *UltraSAN*. In addition, a switch, called the "Low Loss Or Low Delay Switch," for an ATM based network is modeled and cell loss probabilities and average cell delays in the switch are obtained for different types of bursty traffic.

CHAPTER 1
INTRODUCTION

Performance and dependability evaluation of complex systems has emerged as an important area of research. Model-based evaluation requires model construction, specification of performance variables, and model solution. For model-based evaluation of complex systems, an efficient technique is required for model construction and specification of performance variables. A constructed model of a system can be solved for the desired performance variables either by simulation or by analytic solution.

Though simulation is a popular method for solution, it has a couple of drawbacks. First of all, solution by simulation may take an excessively long time for the desired performance variables to converge within a specified confidence interval. Second, simulation may fail to solve for a performance variable which depends upon "rare" events in a system. Analytic solution avoids these problems, but presents challenges of its own. When doing an analytic solution, a stochastic process is generated from the constructed model, and generated stochastic process is solved for the desired performance variables using mathematical techniques.

Petri nets [31] and stochastic extensions [29, 26, 22, 8], called stochastic Petri nets (SPNs), were introduced as a graphical framework for model construction and performance variable specification. Systems modeled using SPNs can be solved by either analysis or simulation. But for analytic solution, the distributions of different

times in the modeled system were restricted to an exponential distribution so that the underlying stochastic process was Markov and could be solved easily. In many realistic systems, some times are deterministic, such as propagation delay in computer networks, or time slots for a scheduled process in operating systems, etc. Since the exponential distribution has very high variance, it might be inaccurate to model a deterministic time (which has zero variance) as an exponentially distributed time.

M. Ajmone Marsan and G. Chiola [23] developed a mathematical technique in deterministic and stochastic Petri nets (DSPNs), which allowed analytic solution of models with exponentially distributed times along with deterministic times. However, the numerical integration technique they used as part of the solution procedure was so slow that it restricted the applicability of the technique to very small problems. Using the same basic idea, Christoph Lindemann [18, 19] applied uniformization instead of numerical integration and achieved significant improvement in efficiency.

This thesis implements the mathematical technique, introduced by M. Ajmone Marsan and G. Chiola, and improved by Christoph Lindemann, for stochastic activity networks (SANs). SANs are mathematically equivalent to SPNs, but have a hierarchical modeling capability were introduced in [34, 25, 27]. SANs have more modeling power than SPNs since they have constructs called "input gates" and "output gates" which help to model the complex mechanics of a system. The next section briefly reviews SANs.

1.1 Stochastic Activity Networks

Structurally, Stochastic activity networks (SANs) [25] consist of "activities," "places," "input gates," and "output gates." All of these constructs are properly connected via

directed arcs to form a *subnet*.

Activities, which are similar to transitions in Petri nets, are of two types: "timed" and "instantaneous." *Timed activities* represent activities of the modeled system whose duration impacts a system's ability to perform. *Instantaneous activities* represent the system activities which, relative to the performance/dependability variables of interest, complete in a negligible amount of time. "Cases" are associated with activities. The probabilities associated with *cases* provide an ability to perform different actions when an activity completes.

Places are similar to the places in Petri nets. They contain tokens. The number of tokens in all places in the network is called a *marking* of the network. The number of tokens in each place is called a marking of that place.

Input gates and *output gates* permit greater flexibility in defining enabling and completion rules for the activities than regular Petri nets. Input gates have "enabling predicates" and "functions." An *enabling predicate* is a boolean function on the marking of the places connected to the input gate. The *function* of an input gate describes an action on the marking of the connected places which will be executed upon the completion of the activity to which the input gate is connected. Output gates also have "functions," which describe an action on the marking of the connected places which will be executed upon the completion of the activity to which the output gate is connected.

Two operations, "join" and "replicate," are defined on the subnets which make hierarchical modeling possible [35]. The *join* operation produces another subnet by combining different subnets, while keeping a subset of the available places common among the joined subnets. The *replicate* operation produces another subnet by rep-

licating a subnet a certain number of times. Thus the replicate operation allows the construction of a composed model which consists of several identical subnets. A complex system can thus be divided into small parts. A subnet can be constructed for each part and then the subnets can be joined and replicated together to construct the composed model of the system.

A few terms related to a SAN execution are now defined. An activity is *enabled* in a marking when each of its input gate's predicate is true, and each of its directly connected input places contains at least one token. A *stable marking* is a marking in which no instantaneous activity is enabled. If at least one instantaneous activity is enabled in a marking of a SAN, it is an *unstable marking*. The time between activation and completion of an activity is called *activity time*. When an activity remains enabled throughout its activity time, it *completes*.

When an activity completes, the marking of each of its directly connected input places is decremented by one, the function of each of its input gate is executed, a case of the activity is chosen, and function of each of the output gate connected to the case is executed. Thus, when a timed activity completes in a stable marking, the SAN executes, and another marking is reached. If any instantaneous activity is enabled in this new marking, it completes immediately and, due to the execution of the SAN, another marking is reached. Thus, a SAN can reach a new stable marking from a given stable marking due to the completion of a timed activity, and zero or more instantaneous activities.

From a given stable marking, more than one next stable marking is possible due to the fact that the timed activity and the instantaneous activities following it may have more than one case, and in a given unstable marking more than one instantan-

eous activities may be enabled. The probability of reaching a particular next stable marking from a given stable marking when a timed activity completes, depends upon the case probabilities of the timed activity and the instantaneous activities following it before reaching the next stable marking. A SAN is *well specified* if, in all reachable stable markings, the probability distribution of all the next possible stable markings is the same. The *well specified check* [34] checks if a SAN is well specified in a given stable marking. It also gives the probabilities of reaching all the next possible stable markings from a given stable marking.

To model and solve large systems, a model construction technique known as "Reduced Base Model Construction" (RBMC) [34, 35] was introduced. The RBMC technique constructs a stochastic process which supports designated performance variables (such stochastic process is called *reduced base model*). The notion of a "state" in SANs is such that there can be many-to-one mapping between markings of a SAN and states of the underlying stochastic process. For example, when a subnet is replicated, RBMC technique does not keep track of a marking of each replica, but it keeps track of the different markings and a number of replicas in them. However, for very simple SANs, there may be one-to-one mapping between markings and states and in that case different markings can be considered as different states. The RBMC technique has been incorporated into *UltraSAN* [6].

1.2 Research Objective

As mentioned earlier, for analytic solution of SANs, the RBMC technique generates a stochastic process (reduced base model). For solution to proceed, one of the

necessary conditions was that the activity times of activities must be exponentially distributed so that the "behavior" of the SAN could be Markov [34]. As mentioned before, deterministic and stochastic Petri nets (DSPNs) were introduced by M. Ajmone Marsan and G. Chiola [23, 24]. In [18, 19] a numerical algorithm for a stable and efficient computation of the steady-state solutions of DSPN models was presented. Having analytic solution of SANs with deterministic activities is useful to model systems with deterministic times.

The goal of this thesis is to:

- Adapt the algorithms for the RBMC technique to allow activities with a deterministic activity time distribution.
- Implement the algorithm for a stable and efficient numerical solution technique in *UltraSAN* for models having activities with exponentially and/or deterministically distributed activity times.
- Illustrate the usefulness of the implemented solution technique by obtaining a solution to a realistic model.

Chapter 2 explains the concepts used to deal with the non-memoryless property of deterministic activity times and the mathematical framework, which gives the exact steps for the solution technique.

Chapter 3 explains the changes made in the existing RBMC technique to allow deterministic activity times and the solution technique as implemented in *UltraSAN*.

Chapter 4 illustrates an application of the implemented solution technique which considers switching in an Asynchronous Transfer Mode-based network.

Chapter 5 presents conclusions reached from the study and suggests possible areas for further research.

CHAPTER 2

THEORY

This chapter reviews a solution technique for stochastic activity networks (SANs) with deterministic activities. The solution technique was developed for DSPNs [23, 18]. The concept on which the mathematical framework is built is presented in Section 2.1, followed by a section on the details of the framework. Finally, a small example illustrating the technique is presented in Section 2.3.

As mentioned in the previous chapter, the notion of a state of a generated stochastic process is such that there can be many-to-one mapping between markings of a SAN and states of the underlying stochastic process. For very simple SANs, there can be one-to-one mapping between states and markings. Thus, for the simplicity of discussion, marking is considered to be the same as a state of the stochastic process.

2.1 The Concept

We will consider a continuous-time Markov process (henceforth a continuous-time Markov process will be referred as a Markov process), which is completely specified by an infinitesimal generator matrix (rate matrix) S and an initial state probability distribution. Once S is specified, the stationary distribution for the state occupancy probability vector π can be found. As shown in [14], when the Markov process is irreducible and aperiodic, the stationary distribution is the same as the limiting

distribution (also called the steady-state distribution) and is independent of the initial state probability distribution. In this case, the steady-state distribution for the state occupancy probability vector π can be found by any of the following methods:

1. Solve the equations:

$$\begin{aligned} \pi \cdot S &= \overline{0} \\ \sum_i \pi_i &= 1. \end{aligned}$$

2. Find the embedded Markov chain subordinated to a Poisson process [12, 11] as follows. Let,

$$\begin{aligned} s &= max \mid S_{ii} \mid \\ \widehat{P} &= \frac{S}{s} + I \end{aligned}$$

where, $\widehat{P}$ is the state transition probability matrix for the subordinated discrete time Markov process (henceforth, a discrete time Markov process will be referred as a Markov chain) subordinated to a Poisson process. Now solve,

$$\begin{aligned} \pi \cdot \widehat{P} &= \pi \\ \sum_i \pi_i &= 1. \end{aligned}$$

3. Solve the equations:

$$\tilde{\pi} \cdot P = \tilde{\pi} \tag{2.1}$$

$$\sum_i \tilde{\pi}_i = 1 \tag{2.2}$$

$$\tilde{\pi} \cdot C = \pi \tag{2.3}$$

where, P is a state transition probability matrix of an embedded Markov chain obtained from the Markov process S by looking only at times of state changes. $\tilde{\pi}$ is the state occupancy probability vector for the Markov chain obtained by looking at the Markov process at times of state changes. C is a conversion matrix which accounts for a sojourn time of the Markov process in a state. This method is used to find an analytical solution of a SAN with deterministic activities.

The next section explains how to find the P and C matrices, when deterministic activities are present.

2.1.1 Solution by Embedded Markov Chain

In this section, the solution by a technique of an embedded Markov chain is explained when all the activities enabled in a marking have only exponentially distributed activity times or there may be, at the most one, deterministic activity concurrently- and/or competitively-enabled with the exponential activities.

2.1.1.1 Embedded Markov Chain with Only Exponential Activities

When activities with only exponentially distributed activity times (exponential activities) are enabled in a marking of a SAN, the behavior of the SAN is a Markov process [34]. In this case, the rows of the P and C matrices can be obtained from the infinitesimal generator matrix S of the Markov process. Let, S_{ij} be the $(i,j)th$ element of S, or, in other words, the rate of a transition from a marking i to a marking

j $(j \neq i)$. Then, the $(i,j)th$ elements of the P and C matrices can be calculated as:

$$\begin{aligned} P_{ij} &= \frac{S_{ij}}{\sum_{x \neq i} S_{ix}} \qquad if\ j \neq i \\ &= 0 \qquad\qquad if\ j = i \end{aligned} \tag{2.4}$$

$$\begin{aligned} C_{ij} &= \frac{1}{\sum_{x \neq i} S_{ix}} \qquad if\ j = i \\ &= 0 \qquad\qquad if\ j \neq i. \end{aligned} \tag{2.5}$$

P_{ij} accounts for the transitions among the markings and C_{ii} is an average sojourn time in the marking i. When there are only exponential activities in a system, this method is not used normally since it involves one extra vector-matrix multiplication then the other two methods, but it provides a basis for understanding solution of SAN with deterministic activity times.

2.1.1.2 Embedded Markov Chain with Deterministic Activities

The concept presented here is based upon DSPNs, which were introduced in [23]. For simplicity, only one activity with a deterministic activity time (deterministic activity) is considered in the discussion, but the concept is general enough to obtain the solution when more than one deterministic activity is in the system with the restriction that no more than one deterministic activity be enabled in a given marking. The solution process, explained below, should be repeated to obtain the rows of the P and C matrices for each deterministic activity.

The deterministic activity under consideration will be called A_T. Let T be a marking-independent activity time of A_T. Figure 2.1 shows a partition of all the possible stable markings of the system. The partition MD is a set of markings in

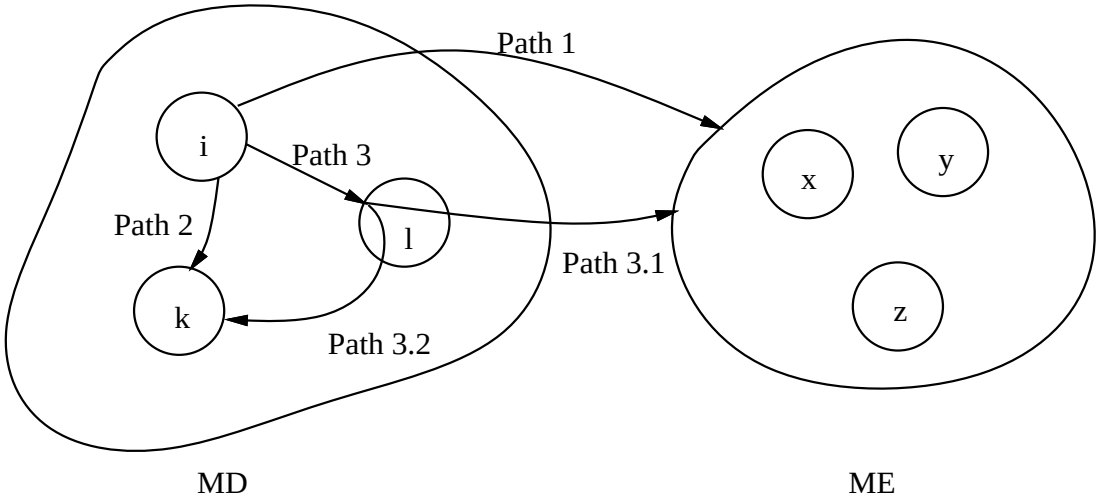

Figure 2.1: Partition of Markings of the System with Deterministic Activities

which the activity A_T is enabled. The partition ME is a set of markings in which the activity A_T is not enabled. It consists of the markings in which only exponential activities are enabled. (In general, other deterministic activities might be enabled in the markings of set ME, but since only one deterministic activity is considered in the system, only exponential activities are considered in the markings of set ME.)

a For the markings in which only exponential activities are enabled, the embedded Markov chain is obtained by looking at the Markov process at times of state changes. But for the markings in which a deterministic activity is enabled, the embedded Markov chain is obtained by looking at times when either the deterministic activity either completes or is aborted. The matrices P and C are obtained for such embedded Markov chain.

The calculation for the rows of the P and C matrices is straight-forward for all the markings in set ME. Since only exponential activities are enabled in all the markings in this set, equations 2.4 and 2.5 can be used directly.

Now consider the markings in set MD. Suppose that the system is in a marking i. In marking i, the activity A_T may be exclusively enabled (i.e. it is the only enabled activity in the marking i), concurrently-enabled with the other exponential activities (i.e. completion of any concurrently-enabled exponential activity does not disable A_T), or competitively-enabled with the other exponential activities (i.e. completion of any competitively-enabled exponential activity disables A_T). Each of these cases is considered in the following paragraphs.

Exclusively-enabled Deterministic Activity

As shown by path 1 or path 2 in Figure 2.1, when an exclusively enabled deterministic activity completes, the system can reach a marking either in set ME or set MD. In this case, a row of the P matrix (P_{ij}, $\forall j$) can be found from the case probabilities of the deterministic activity and the case probabilities of the instantaneous activities following the deterministic activity. Since the system spends time T in the marking i, a row of the C matrix can be found as,

$$\begin{aligned} C_{ij} &= T \quad if\ j = i \\ &= 0 \quad else. \end{aligned}$$

Deterministic Activity, Concurrently- or Competitively-enabled with Exponential Activities

When a concurrently-enabled exponential activity completes before the deterministic activity, the system remains in one of the markings in set MD and the deterministic activity is still enabled. Alternatively, when a competitively-enabled exponential

activity completes before the deterministic activity, the system reaches one of the markings in set ME and the deterministic activity is aborted.

To be more specific, consider path 3 in Figure 2.1. After starting from marking i, suppose the system is in a marking l due to the completion of one or more concurrently-enabled exponential activities. If the deterministic activity A_T completes in marking l, the system reaches any marking either in set ME or MD. If a competitively-enabled exponential activity completes before A_T in marking l, the system follows path 3.1 and reaches a marking in set ME and A_T is aborted. If a concurrently-enabled exponential activity completes before A_T in marking l, the system follows path 3.2 and reaches a marking k in set MD and A_T is still enabled.

Recall that the system is observed only at the completion or abortion of the deterministic activity A_T. This means that an embedded Markov chain is obtained by looking at the completion or abortion of the deterministic activity A_T, when the system is in a marking in set MD, or looking at the time of transition, when the system is in a marking in set ME. The Markov chain obtained in this fashion is called an embedded Markov chain subordinated to a deterministic activity [18]. The probability of reaching any marking w (in either of the sets) from the marking i is found when either A_T completes or aborts, which gives the ith row of the P matrix. The steady-state state occupancy probability vector is then obtained by solving a system of linear equations as in equation 2.1.

The result obtained in this fashion, however, does not represent the state occupancy probabilities of the system. Consider the following scenario. Initially, the system is in marking i. It reaches a marking w (when A_T completes or aborts) via marking $l \in MD$. After starting from marking i, the system visits marking l due to

the completion of an exponential activity concurrently-enabled with A_T. But since the system is observed only at the completion or abortion of A_T, the embedded Markov chain subordinated to a deterministic activity does not visit marking l after starting from marking i. While solving equation 2.1 it was assumed that before reaching marking w, the systems spends all of the time T in marking i.

However, the system may spend some portion of time T in marking l as well. So, the sojourn time in markings i and l should be calculated when i is the initial marking and the steady-state probability that the system is in the marking i should be redistributed between the markings i and l proportional to the sojourn time in each marking. This redistribution is achieved with the help of the C matrix (conversion matrix). When the steady-state state occupancy probability vector, as obtained in equation 2.1, is multiplied with the C matrix, as in equation 2.3, all the steady-state state occupancy probabilities are redistributed properly according to the sojourn times. So, the ith row of the C matrix can be calculated from these sojourn times. In this case, C_{il} = the sojourn time in the marking l given that the system is in an initial marking i.

In general, the sojourn time in all markings is set MD due to the completion of the exponential activities should be calculated considering i as an initial marking. The steady-state probability of being in marking i should be redistributed among all these markings proportional to their sojourn times. These sojourn times forms the ith row of the C matrix.

The method to find the rows of the P and C matrices, as explained above, should be repeated for all the markings in set MD, considering that marking as an initial marking. The procedure should be repeated for all the deterministic activities. Only

there after, the rows of the P and C matrices should be calculated for the markings in which no deterministic activities are enabled using equations 2.4 and 2.5.

The following section explains the mathematical procedure used to calculate the rows of the P and C matrices, using the method given in [23].

2.2 Mathematical Framework

In this section, the solution technique to find the rows of the P and C matrices when a deterministic activity A_T is concurrently- or competitively-enabled with the other exponential activities is presented. The technique is explained considering a single deterministic activity in the system, and should be repeated for all the deterministic activities in the system. Let,

A_T = a deterministic activity under consideration,
T = a marking independent activity time of A_T,
N = the total number of the stable markings in the system,
MD = a set of markings in which A_T is enabled,
ME = a set of markings in which A_T is not enabled,
M_k = a marking k,
P = the probability transition matrix of the embedded Markov chain,
C = the conversion matrix,
Q = the rate matrix due to the concurrent and competitive exponential activities[1],
Q_{ij} = an ith row, jth column element of Q,
π = the steady-state state occupancy probability vector,
π_i = an ith element of π,
$\overline{u_x}$ = the row vector with only xth element 1, the rest are 0,
$\overline{u_x'}$ = the column vector with only xth element 1, the rest are 0.

The following section shows how to calculate a row of the P matrix, followed by a section showing a method to calculated a row of the C matrix when a deterministic activity is enabled in a marking.

[1]For the markings in which A_T is enabled, rows of the matrix Q are the same as the matrix S. All other rows of the matrix Q are zero.

2.2.1 Solution for the Probability Transition Matrix (P)

The probability of reaching a marking j from an initial marking i must be calculated when either A_T aborts or completes. Then, using the total probability theorem, a row of the P matrix is found. These two cases, when A_T is aborted and when A_T completes, are considered in the following paragraphs.

When A_T is Aborted

Suppose marking i (M_i) is the initial marking. Since A_T is aborted, the resulting marking j (M_j) must be in set ME. Thus, $\forall M_i \in MD,\ M_j \in ME$,

$$Prob\{M_i \rightarrow M_j,\ A_T\ aborted\} =$$

$$\int_{t=0}^{T} \left[\sum_{\forall M_k \in MD} (P\{M_k \rightarrow M_j\ at\ time\ t\} \cdot P\{M_k\ at\ time\ t \mid M_i\ at\ time\ 0\}) \right] dt.$$

The second probability in the equation indicates the probability of being in a marking k at time t after starting from an initial marking i due to the exponential activities just before the abortion of A_T. Obviously, M_k is in set MD. The first probability indicates the probability of a transition from $M_k \in MD$ to a marking j due to the completion of a competitively-enabled exponential activity. Obviously M_j is in set ME and A_T is no longer enabled in M_j, i.e. A_T is aborted.

At time t, just before the abortion of A_T, the system can be in any of the markings in set MD. The total probability is thus obtained by summing over all the possibilities, i.e. summing over all the possible $M_k \in MD$. The time t at which the abortion of A_T occurs varies from 0 to T, which is the range of integration.

Using the Chapman-Kolmogorov (C-K) equation [15], this equation can be written

as follows. $\forall M_i \in MD,\ M_j \in ME$,

$$Prob\{M_i \rightarrow M_j,\ A_T\ aborted\} = \int_{t=0}^{T} \left[\sum_{\forall k: M_k \in M_D} \left(\left(\overline{u_i} \cdot e^{Qt} \right)_k (Q_{kj} dt) \right) \right].$$

The first term, $\left(\overline{u_i} \cdot e^{Qt} \right)_k$, in the summation gives the probability of reaching a marking k from the initial marking i ($\overline{u_i}$ is a row vector, as defined above) at time t due to completion of one or more exponential activities (indicated by the rate matrix Q). If the subscript k is dropped from the first term, then it gives a row vector of probabilities of reaching any marking at time $t \leq T$, given that the system is in marking i initially, by completion of one or more exponential activities. The second term gives the probability of reaching a marking j from marking k due to the completion of a competitively-enabled exponential activity at time t. Since for $M_x \notin MD$, a row in Q matrix is $\overline{0}$ (i.e. $Q_{xj} = 0\ \forall j$ if $M_x \notin MD$), using vector notation this equation can be written as follows. $\forall M_i \in MD,\ M_j \in ME$,

$$\begin{aligned} Prob\{M_i \rightarrow M_j,\ A_T\ aborted\} &= \int_{t=0}^{T} (\overline{u_i} \cdot e^{Qt}) \begin{bmatrix} Q_{1j} dt \\ Q_{2j} dt \\ \vdots \\ Q_{Nj} dt \end{bmatrix} \\ &= \int_{t=0}^{T} (\overline{u_i} \cdot e^{Qt})(Q \cdot \overline{u_j'}) dt \end{aligned}$$

where, $Q \cdot \overline{u_j'}$ gives jth column of the Q matrix. After integration, $\forall M_i \in MD,\ M_j \in ME$,

$$\begin{aligned} Prob\{M_i \rightarrow M_j,\ A_T\ aborted\} &= \overline{u_i} \cdot \left(e^{QT} - I \right) \cdot \overline{u_j'} \\ &= \overline{u_i} \cdot e^{QT} \cdot \overline{u_j'}. \end{aligned}$$

Representing this equation in a vector form to find the ith row of the P matrix when A_T is aborted,

$$P_{i-,A_T \ aborted} = \overline{u_i} \cdot e^{QT} \cdot J. \tag{2.6}$$

In equation 2.6, J is an $N \times N$ matrix consisting of the columns vectors $\overline{u_j'}$, $\forall M_j \in ME$. Mathematically,

$$\begin{aligned} J_{ij} &= 1 \quad if\ j = i\ and\ M_j \in ME \\ &= 0 \quad else. \end{aligned} \tag{2.7}$$

Equation 2.6 gives an ith row of the P matrix when A_T is aborted due to a competitively-enabled exponential activity. In the following paragraph, the probability of reaching a marking j from an initial marking i when A_T completes is obtained which gives an ith row of the P matrix when A_T.

When A_T Completes

Since A_T completes, the resulting marking can be in either of the two sets. Suppose the system is in a marking x just before the completion of A_T and when A_T completes, the system jumps to another (possibly unstable) marking y. The system can reach a marking z from marking y due to zero or more instantaneous activities. Thus, after completion of A_T, system can reach one of the different possible markings and the probability of reaching different markings from the marking x depends upon the case probabilities of A_T and the instantaneous activities following A_T. Let D' be the transition probability matrix due to the completion of A_T, and the instantaneous activities following it, for the markings in set MD. For markings in the set ME, the

rows of D' are $\overline{0}$. Mathematically,

$$\begin{aligned} D'_{xj} &= 0 \quad if\ M_x \in ME \\ &= d_{xj} \quad else \end{aligned} \tag{2.8}$$

where d_{xj} can be found from the case probabilities of A_T and the instantaneous activities following it [34]. Thus, $\forall\ M_i \in\ MD$, and an M_j,

$$Prob\{M_i \rightarrow M_j,\ A_T\ completed\} = \overline{u_i} \cdot e^{QT} \cdot D' \cdot \overline{u'_j}.$$

$\overline{u_i} \cdot e^{QT}$ gives the transient state probability vector at time T, just before the completion of A_T, given that the system is in a marking i initially. When it is multiplied with D', it gives the transient state probability vector at time T after the completion of A_T and the instantaneous activities following it given that the system is in a marking i initially. The jth element of this vector is obtained by multiplying it with $\overline{u'_j}$. The vector form of this equation gives an ith row of the P matrix when A_T completes. Thus,

$$P_{i-,A_T\ completes} = \overline{u_i} \cdot e^{QT} \cdot D'. \tag{2.9}$$

A Row of P matrix

According to the total probability law and Bayes' theorem, the addition of equations 2.6 and 2.9 gives the probability of going from a marking i to any marking, i.e. the ith row, P_{i-}, of the P matrix. So from equations 2.6 and 2.9,

$$\begin{aligned} P_{i-} &= \overline{u_i} \cdot e^{QT} \cdot J + \overline{u_i} \cdot e^{QT} \cdot D' \\ &= \overline{u_i} \cdot e^{QT} \cdot (J + D') \\ &= \overline{u_i} \cdot e^{QT} \cdot D \end{aligned} \tag{2.10}$$

where D can be defined from equations 2.7 and 2.8 as

$$\begin{aligned} D_{ij} &= d_{ij} \quad if\ M_i \in MD \\ &= 1 \quad if\ j = i\ and\ M_i \in ME \\ &= 0 \quad else. \end{aligned}$$

Thus, D is the transition probability matrix due to the completion of A_T and instantaneous activities following it. If A_T is not enabled in a particular marking, then the corresponding row in D matrix is 0 except at the diagonal where it is 1.

Equation 2.10 is the same as obtained in [23]. The matrix exponent can be calculated using uniformization[18], as shown in Section 2.2.3. The next section shows a method to calculate a row to the conversion matrix the C for a marking in which a deterministic activity is concurrently- and/or competitively-enabled with exponential activities.

2.2.2 Solution for the Conversion Matrix (C)

In markings in which a deterministic activity is enabled, the system is observed only at the completion or the abortion of the deterministic activities. A correction must thus be made to the steady-state state occupancy probabilities according to the sojourn time in the different markings during an activity time of the deterministic activity.

Assuming that the system is initially in marking i, the probability of being in marking j at time t is found. Integrating this probability from 0 to T gives the sojourn time in marking j during the activity time of A_T given that the system is

initially in marking i. This forms the (i,j)th element of the C matrix. Thus,

$$C_{ij} = \int_{t=0}^{T} \left(\overline{u_i} \cdot e^{Qt}\right) \cdot \overline{u_j^{'}} dt.$$

In the vector form, the ith row can be written as,

$$C_{i-} = \int_{t=0}^{T} \overline{u_i} \cdot e^{Qt} dt. \tag{2.11}$$

Equation 2.11 is the same as shown in [23]. The matrix exponent is also calculated using uniformization [18], as is shown in the following section.

2.2.3 Uniformization

Uniformization is a well known technique to compute the matrix exponent for the transient analysis of a Markov process [12, 11]. This technique is used here to compute the matrix exponent in the equations 2.10 and 2.11. In this technique, an embedded Markov chain is obtained by subordinating the Markov process with a Poisson process. Consider the transition rate matrix Q of the Markov process. Let,

$$q = 1.02 \cdot max \mid Q_{ii} \mid, \; and \tag{2.12}$$

$$\widehat{P} = \frac{Q}{q} + I. \tag{2.13}$$

A multiplicative factor 1.02 in the definition of q in equation 2.12 ensures that the embedded Markov chain with the transition probability matrix $\widehat{P}$ is aperiodic [38]. The matrix Q can be written as,

$$Q = q(\widehat{P} - I).$$

So,

$$e^{QT} = e^{(\widehat{P}-I)qT} = e^{\widehat{P}qT} \cdot e^{-IqT} = e^{\widehat{P}qT} \cdot I \cdot e^{-qT} = e^{\widehat{P}qT} \cdot e^{-qT}.$$

The matrix exponent in the transient analysis can thus be found as in [12, 11] to be,

$$\begin{aligned} \overline{u_i} \cdot e^{QT} &= \overline{u_i} \cdot e^{\widehat{P}qT} \cdot e^{-qT} \\ &= \sum_{s=0}^{\infty} \overline{u_i} \cdot \frac{(\widehat{P}qT)^s}{s!} \cdot e^{-qT} \\ &= \sum_{s=0}^{\infty} \overline{u_i} \cdot (\widehat{P})^s \cdot \frac{e^{-qT}(qT)^s}{s!}. \end{aligned} \tag{2.14}$$

Now let, $\beta_{qT}(s)$ = probability of s arrivals in a Poisson process with rate q in time $T = \frac{e^{-qT}(qT)^s}{s!}$, and $\pi(s)$ = the transient state occupancy probability vector after s transitions. So,

$$\pi(s+1) = \pi(s) \cdot \widehat{P} \tag{2.15}$$

$$\pi(s) = \pi(0) \cdot (\widehat{P})^s.$$

$\widehat{P}$ is the transition probability matrix of an embedded Markov chain subordinated to a Poisson process with parameter qT. Since, $\overline{u_i} = \pi(0)$, equation 2.14 can be written as,

$$\overline{u_i} \cdot e^{QT} = \sum_{s=0}^{\infty} \pi(s) \cdot \beta_{qT}(s).$$

Using this equation, equation 2.10 can be written as

$$P_{i-} = \overline{u_i} \cdot e^{QT} \cdot D = \left(\sum_{s=0}^{\infty} \pi(s) \cdot \beta_{qT}(s) \right) \cdot D.$$

To truncate the infinite series, a small error tolerance $\epsilon > 0$ is introduced. The left and right truncation points, L and R respectively, are found as in [9] such that,

$$\sum_{s=0}^{L-1} \beta_{qT}(s) < \frac{\epsilon}{2}$$

and

$$1 - \sum_{s=0}^{R} \beta_{qT}(s) < \frac{\epsilon}{2}.$$

Using these left and right truncation points and considering the fact that the maximum possible value for the elements of $\pi(s)$ is 1, the upper bound on error in each element of $P(i)$ is ensured to be less than ϵ. So with the ϵ upper bound on error in each element,

$$P_{i-} = \left(\sum_{s=L}^{R} \pi(s) \cdot \beta_{qT}(s) \right) \cdot D. \tag{2.16}$$

This equation is the same as in [18] and gives a method to compute a row of the P matrix.

A method to calculate a row of the C matrix is now considered. Using uniformization, equation 2.11 can be written as,

$$\begin{aligned} C_{i-} &= \int_{t=0}^{T} \overline{u_i} \cdot e^{\widehat{P} qt} \cdot e^{-qt} dt \\ &= \int_{t=0}^{T} \overline{u_i} \cdot \sum_{s=0}^{\infty} \frac{\widehat{P}^s (qt)^s}{s!} e^{-qt} dt \\ &= \sum_{s=0}^{\infty} \overline{u_i} \cdot \widehat{P}^s \left(q^s \int_{t=0}^{T} \frac{t^s}{s!} e^{-qt} dt \right). \end{aligned} \tag{2.17}$$

Integrating by parts,

$$\int_{t=0}^{T} \frac{t^s}{s!} e^{-qt} dt = \frac{1}{q^{s+1}} - e^{-qT} \sum_{m=0}^{s} \frac{T^m \cdot q^{m-s-1}}{m!}.$$

Using this equation in equation 2.17, $C(i)$ can be written as,

$$C_{i-} = \frac{1}{q} \sum_{s=0}^{\infty} \overline{u_i} \cdot (\hat{P})^s \cdot \left(1 - \sum_{m=0}^{s} \frac{e^{-qT}(qT)^m}{m!}\right).$$

Finally, using the left and right truncation, the definitions of $\pi(.)$, and $\beta_{qT}(.)$,

$$C_{i-} = \frac{1}{q} \sum_{s=0}^{R} \pi(s) \cdot \left(1 - \sum_{m=L}^{s} \beta_{qT}(m)\right). \tag{2.18}$$

This equation is the same as in [18] and gives a method to compute a row of the C matrix.

Thus, using uniformization, the row of the P and C matrices can be calculated. Using proper initial state probability vector ($\pi(0) = \overline{u_i}$), the rows for the P and C can be calculated for each marking i in the set MD. The calculation should be repeated for each deterministic activity.

Thus, the rows of the P and C matrices cab be calculated using equations 2.16 and 2.18 for rows in which a deterministic activity is enabled, or using equations 2.4 and 2.5 for rows in which only exponential activities are enabled. After obtaining matrices P and C, the steady-state probability vector $\tilde{\pi}$ for the embedded Markov chain can be obtained by solving equations 2.1 and 2.2. The steady-state probability vector π can be obtained be solving equation 2.3. The following section illustrates use of this solution technique by small example.

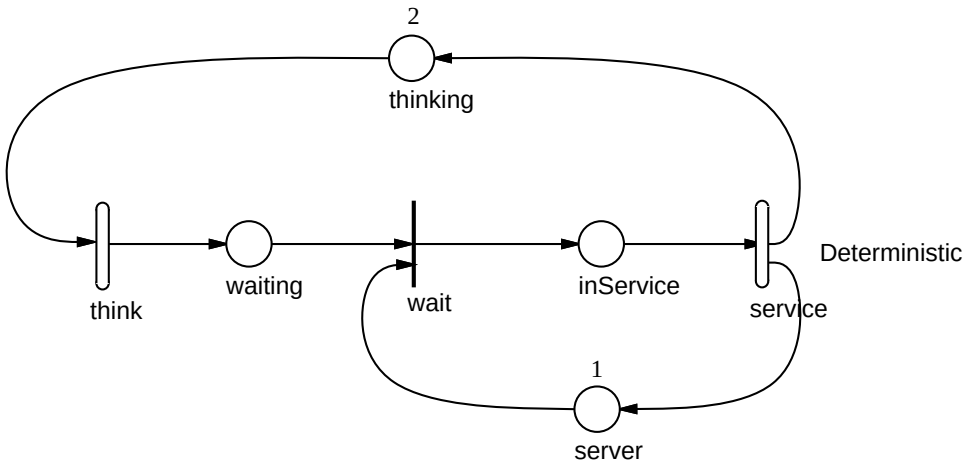

Figure 2.2: SAN Subnet for M/D/1/2 Queue

Table 2.1: Initial Non-zero Markings for SAN Subnet for M/D/1/2 Queue

Place	***Marking***
server	1
thinking	2

2.3 Example

An M/D/1/2 queue, as in [23], is now considered to illustrate use of the mathematical framework discussed in the previous section. An *UltraSAN* model for an M/D/1/2 queue is shown in Figure 2.2. Initial non-zero markings are shown in Table 2.1 and activity time distributions are shown in Table 2.2. The marking of place *thinking* represents number of customers waiting outside the queue. Before entering the queue, time is spent in activity *think* by each customer. *think* is an exponential activity with a marking dependent rate (see Table 2.2). A customer, upon entering, must wait in place *waiting* if the server is not available. A token in place *server* indicates the availability of a server. When the server is available and at least one

Table 2.2: Activity Time Distributions for SAN Subnet for M/D/1/2 Queue

Activity	***Distribution***	***Parameter values***
service	*deterministic*	
	value	T
think	*exponential*	
	rate	MARK(thinking) $* \mu$
wait	*instantaneous*	

customer is in *waiting*, it immediately obtains service. Activity *service* is deterministic. When *service* completes, a server is made available by putting a token back in *server* and the served customer is put in the thinking state by incrementing the marking of *thinking*.

Figure 2.3 shows the possible stable markings of this queue. A label on an arrow shows a name of an activity that causes this transition. Note that the deterministic activity *service* is enabled in markings 2 and 3. In marking 2, both the timed activities are enabled. Since the system is observed only at the completion or abortion of the deterministic activity, the embedded Markov chain subordinated to the deterministic activity will never visit marking 3. So if the P matrix is obtained and equation 2.1 is solved, the steady-state state occupancy probability vector, $\tilde{\pi}$, of the embedded Markov chain can be obtained in the following form.

$$\tilde{\pi} = [p_1,\ p_2,\ 0]$$

where p_1 and p_2 depend upon the rate μ of the exponential activity *think* and the activity time T of the deterministic activity *service* (refer to [23] for the exact values). If c is a fraction of the sojourn time in marking 2 and $1-c$ is a fraction of the sojourn time in marking 3 during the activity time of *service* when the system is initially in

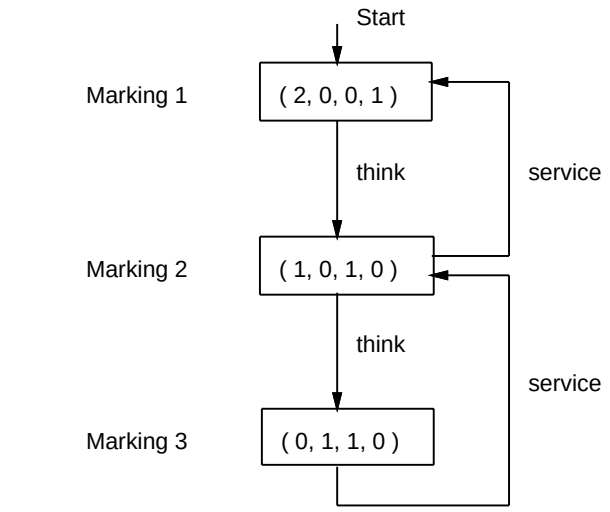

Figure 2.3: Possible Stable Markings of M/D/1/2 Queue

the marking 2, then the C matrix can be obtained as,

$$C = \begin{bmatrix} \frac{1}{2*\mu} & 0 & 0 \\ 0 & cT & (1-c)T \\ 0 & 0 & T \end{bmatrix}$$

Using equation 2.3, and normalizing to 1 as in equation 2.2, the steady-state state occupancy probability vector π can be obtained as follows.

$$p_{123} = \frac{p_1}{2\mu} + p_2 \cdot T$$

$$\pi = \frac{1}{p_{123}} \left[\frac{p_1}{2\mu}, \;\; p_2 cT, \;\; p_2(1-c)T \right].$$

The next chapter discusses the issues related to the implementation of this method in *UltraSAN*.

CHAPTER 3
IMPLEMENTATION

The mathematical procedure presented in the previous chapter to solve SANs with deterministic activities has been implemented in *UltraSAN* [6]. Figure 3.1 shows organization of different modules in *UltraSAN*. At the highest level in *UltraSAN*, editors are used for model specification. A specified model can be solved using simulation or analytically. For analytical solution, a reduced base model is first constructed. To solve SAN with deterministic activities, a new solver module is added to the package and some changes are made in reduced base model construction (RBMC) procedure. The modules which are augmented or added are highlighted in the figure.

Section 3.1 explains changes made to RBMC procedure. Section 3.2 explains the solution procedure based upon the mathematical procedure presented in the previous chapter. Section 3.3 describes a method to improve the speed of the solution procedure.

3.1 Changes in Reduced Base Model Construction Technique

The existing reduced base model construction (RBMC) technique, as implemented in *UltraSAN* [32], generates all the stable markings and the rate of transition from one marking to another when all timed activities in the model are exponentially distributed. As shown in equations 2.10 and 2.11, for SAN with deterministic activ-

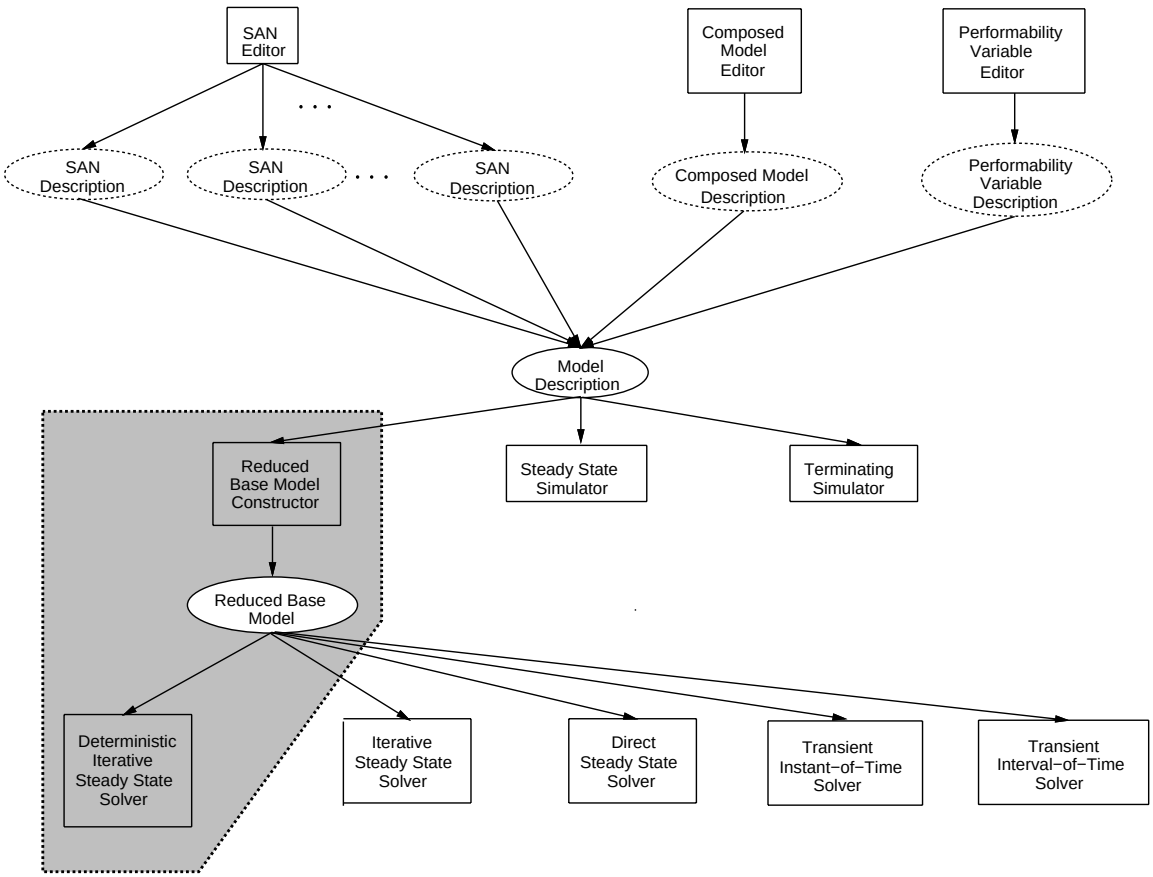

Figure 3.1: Organization of *UltraSAN*

ities, the probabilities of transition from one marking to the other markings are also needed when a deterministic activity is enabled in a marking.

In the existing RBMC technique, a linked list of the next possible stable markings and the rates to these markings is maintained for each stable marking using the structure *RateList* (see Appendix A). In the new implementation, either the rates or the probabilities to the next possible markings are needed. To maintain the compatibility with the existing implementation, the same data structure is used. But as shown in Section A.1, the element *rate* of the structure *RateList* has two interpretations. If it is positive, then it is a rate to another marking. If it is negative, then it is the negative of a probability of reaching another stable marking.

Moreover, an array of all the deterministic activities in the system is maintained. For each deterministic activity, a linked list of all the markings in which it is enabled is kept. See the structures *DetActs* and *StateInDetAct* in Section A.1. These lists (one for each deterministic activity) are stored in *project_name.det* file and used during the solution procedure.

In summary, the following changes were made to the existing RBMC procedure:

1. Along with the exponential activities, the deterministic activities with marking-independent activity times are allowed.

2. While processing all the activities in a given marking, the new RBMC technique checks that only one deterministic activity is enabled in each reachable stable marking.

3. While processing a particular case of an activity in a marking (a current marking), the **WellSpecifiedCheck** algorithm [32] gives all the next possible stable

markings and the probabilities to them from the current marking. If the activity under consideration is exponential, no change in the existing algorithm is required. The rates to the next stable markings are found by multiplying the probabilities to the rate of the exponential activity and added to the *RateList* for the current marking. If the activity is deterministic, the probabilities to all the next stable markings are added to the *RateList* for the current marking. Thus from a current marking, the system can reach the other markings by the completion of the exponential activities and/or a deterministic activity. In the first case the rate is included and in the later case the probability is included in the linked list of the next stable markings (*RateList*) for the current marking. It might be possible that in the *RateList* for a given current marking, another marking appears twice, either due to completion of a deterministic activity or due to completion of one or more exponential activities.

4. While processing an activity in a marking, if the activity is deterministic, then this marking is added in the *StateInDetAct* list for this deterministic activity.

The following section shows how to form the system of linear equations from the generated reduced base model, and solve them using the procedure in the previous chapter.

3.2 Deterministic Iterative Steady State Solver

In the solution procedure, rows for the P and C matrices are calculated in a loop for all stable markings. If a deterministic activity is enabled in a marking, then the appropriate rows of the P and C matrices are found by calling procedure *genRowPiCi*.

If only exponential activities are enabled in a marking, then the rows of the P and C matrices are found by calling procedure *findPiCi*. At the end of the loop, a system of linear equations is solves using matrix P by calling procedure *solveP* and a correction is applied using matrix C by calling procedure *correction*.

Section 3.2.1 explains the main loop of the solution procedure. Section 3.2.2 explains an algorithm to find the rows of P and C matrices when a deterministic activity is enabled in a marking. Section 3.2.3 explains an algorithm to find rows of P and C matrices when only exponential activities are enabled in a marking. Section 3.2.4 shows how a system of linear equations is solved iteratively.

3.2.1 Solution

The new RBMC technique generates all possible stable markings, the rate of transition from one marking to another marking due to the exponential activities and the probability of transition from one marking to another marking due a deterministic activity. This information is stored in the appropriate files by the RBMC technique. The following matrices are reconstructed from the files and are input to the solution procedure.

S = the transition rate matrix, and

V = the transition probability matrix.

The reward structure is also input to the solution procedure which is used to calculate the required performance variables when the steady-state state occupancy probability vector is obtained.

The major task of the new solver is to compute the rows of the probability transition matrix P and the conversion matrix C. As discussed earlier, the rows of the P

and C matrices are calculated differently when a deterministic activity is enabled in a marking or all the enabled activities in a marking are exponential.

For each deterministic activity, the transition rate matrix (Q) due to the concurrently- and competitively-enabled exponential activities and the transition probability matrix (D), due to the completion of the deterministic activity and instantaneous activities following it, is constructed. A row of the P and C matrices is then found using the routine **genRowPiCi** for all the markings in which the deterministic activity is enabled, with that marking as an initial marking.

For all the other markings in which only exponential activities are enabled (and so not considered yet), the rows of the P and C matrices are found using the routine **findPiCi**.

The system of linear equations is solved in the routine **solveP** to obtain the steady-state state occupancy probability vector. This vector is multiplied with the conversion matrix C in the routine **correction** which gives the actual steady-state state occupancy probabilities.

Using the steady-state state occupancy probabilities, the performance variables are calculated. The algorithm is summarized as follows.

Procedure 3.2.1 *:* **Deterministic Iterative Steady State Solver (diss)**

Input:
S, V as mentioned above, and reward structure.

Variables:
- Q = the transition rate matrix due to exponential activities concurrently- or competitively-enabled with a deterministic activity.
- D = the transition probability matrix due to the completion of the deterministic activity and the instantaneous activities following it.
- T = activity time of the deterministic activity under consideration.

Begin:

$\forall$ deterministic activities

T = activity time of the selected deterministic activity.

/* Build Q and D matrices for this deterministic activity */
$\forall$ marking i in which this deterministic activity is enabled
ith row of Q matrix = ith row of S matrix
ith row of D matrix = ith row of V matrix

All the other rows of Q matrix = $\overline{0}$
All the other rows of D matrix = 1 at the diagonal, otherwise 0

$\forall$ marking i in which this deterministic activity is enabled
genRowPiCi()

$\forall$ marking i in which no deterministic activity is enabled
findPiCi()

/* solve $\pi' \cdot P = \pi'$ as in equation 2.1 */
solveP()

/* find $\pi = \pi' \cdot C$ as in equation 2.3 */
correction()

Calculate the required performance variables using π and reward structure

End.

3.2.2 genRowPiCi()

genRowPiCi() calculates the rows of P and C matrices for a marking i in which a deterministic activity with an activity time T is enabled, using the uniformization technique presented in the previous chapter. The marking i is the initial marking for the transient analysis.

The Poisson parameter λ is found by multiplying T with 1.02 times the absolute value of the maximum diagonal element of the rate matrix Q.

Since T is non-zero, $\lambda = 0$ indicates that the deterministic activity under consideration is exclusively enabled (no exponential activities are enabled). In this case, the ith row of the P matrix is equal to the ith row of D matrix. The system spends time T in marking i before the completion of the deterministic activity, which is the same as the time spent by the embedded chain in marking i. So the diagonal element of the ith row of the C matrix is T and the other elements of the same row are 0.

When λ is not zero, the ith row of the C and P matrices are calculated using equations 2.16 and 2.18. For calculating the Poisson probabilities, left and right truncation points for a given λ, and error bound ϵ, the algorithm given in [9] is employed, which ensures that the machine underflow and overflow do not occur and has the modest work and space requirements. A single-step transient state occupancy probabilities are calculated iteratively. Using these probabilities and Poisson probabilities, the ith row of the P and C matrices can be found as shown in the following algorithm (where $\oplus$ means the vector addition and $\times$ means the vector-matrix multiplication).

Procedure 3.2.2 *:* **genRowPiCi**

Input:
Q, D, T = the variables as defined in procedure 3.2.1
i = ith row of the matrix.

Variables:
q = the maximum diagonal element of Q
$\widehat{P}$ = the transition probability matrix of an embedded Markov chain subordinated to a Poisson process.
λ = Poisson parameter.

Begin:

Find q as in equation 2.12
Find $\widehat{P}$ as in equation 2.13
Let $\lambda = (1.02 \cdot q)T$

if $\lambda = 0$
ith row of P matrix = ith row of D matrix
ith row of C matrix = T at the diagonal otherwise 0

else
Find the left and right truncation points L and R for a given λ
Find Poisson probabilities $\beta_{qT}(s)$ for s ranging from L to R

$sum = 0$
$tmpP = \overline{0}$ (zero vector)
$tmpC = \overline{0}$ (zero vector)
$\pi(0) = \overline{u_i}$

For $v = 0$ *to* R

if $v < L$
$tmpC = tmpC \oplus \pi(v)$

else
$sum = sum + \beta_{qT}(v)$
$tmpC = tmpC \oplus (1 - sum)\pi(v)$
$tmpP = tmpP \oplus sum \cdot \pi(v)$

$\pi(v+1) = \pi(v) \times \widehat{P}$

$P_{i-} = tmpP \times D$

$C_{i-} = \frac{tmpC}{q}$

End.

3.2.3 findPiCi()

When no deterministic activities are enabled in a marking, the rows of the P and C matrices can be found from the rates of transitions from the marking as shown

below.

Procedure 3.2.3 *:* **findPiCi**

Input:
Q = the variable as defined in procedure 3.2.1
i = ith row of the matrix.

Variables:
λ_{ij} = a rate from a marking i to a marking j due to the exponential activities when the deterministic activity under consideration is enabled.

Begin:

Let $\lambda_{ij} = Q_{ij}$

Find the ith row of the P matrix as shown in equation 2.4
Find the ith row of the C matrix as shown in equation 2.5

End.

3.2.4 solveP()

Once the P matrix is obtained, the system of linear equations is solved as shown in equation 2.1. For this purpose, the Successive Over Relaxation (SOR) method [37, 39], which solves the system of linear equations iteratively, is used. In this method, the following equations are solved:

$$\tilde{\pi} \cdot A = B$$

$$\sum_i \tilde{\pi}_i = 1.$$

The first equation has many possible solutions. One solution is obtained by an iterative method, in which the following equation is solved in the $k+1$st iteration using

the values obtained in the kth iteration:

$$\tilde{\pi}_i^{k+1} = \frac{w}{A_{ii}} \left(B_i - \sum_{j=1}^{i-1} A_{ij} \tilde{\pi}_j^{k+1} - \sum_{j=i+1}^{N} A_{ij} \tilde{\pi}_j^{k} \right) + (1-w)\tilde{\pi}_i^{k} \tag{3.1}$$

where w is the chosen weight factor. Equation 3.1 is iterated until n iterations so that $\| \tilde{\pi}(n) - \tilde{\pi}(n-1) \| <$ specified convergence accuracy (α). Then the second equation is solved by normalizing the obtained solution to one.

Since, $\tilde{\pi} \cdot \widehat{P} = \tilde{\pi}$ is equivalent to $\tilde{\pi} \cdot (\widehat{P} - I) = 0$ equation 3.1 is solved with $B = \overline{0}$ and $A = \widehat{P} - I$. To maintain the compatibility with the existing iterative steady state solver (iss), the data structure for the sparse matrix representation as in [37] is used.

3.3 Steady State Detection

When the Poisson parameter λ is not zero, the rows of the P and C matrices are obtained by the transient analysis of an embedded Markov chain using uniformization. The number of uniformization problems to be solved is equal to the number of markings in which a deterministic activity is enabled. The purpose of steady-state detection is to speed up the solution of each uniformization problem. As shown in the algorithm **genRowPiCi()**, the transient state occupancy probability vector $\pi(v)$ at time v, v ranging from 1 to R, is calculated iteratively using vector-matrix multiplication. Depending upon the behavior of the modeled system, the "steady state" can be reached quite before the Rth iteration and hence fewer vector-matrix multiplications are needed. If the steady state is detected in the iteration $S < R$, then $\pi(S+1)$ approximately equals to $\pi(S)$, and so on. In this case, equation 2.16 can be written

as:

$$\begin{aligned} P_{i-} &= \left(\sum_{s=L}^{S} \pi(s) \cdot \beta_{qT}(s) + \sum_{s=S+1}^{\infty} \pi(S) \cdot \beta_{qT}(s)\right) \cdot D \\ &= \left(\sum_{s=L}^{S} \pi(s) \cdot \beta_{qT}(s) + \pi(S) \sum_{s=S+1}^{\infty} \beta_{qT}(s)\right) \cdot D \\ &= \left(\sum_{s=L}^{S} \pi(s) \cdot \beta_{qT}(s) + \pi(S)\left(1 - \sum_{s=0}^{S} \beta_{qT}(s)\right)\right) \cdot D. \end{aligned}$$

Similarly, equation 2.18 can be written as,

$$C_{i-} = \frac{1}{q}\left[\sum_{s=0}^{S} \pi(s) \cdot \left(1 - \sum_{m=L}^{s} \beta_{qT}(m)\right) + \pi(S)\left(\sum_{s=S+1}^{R}\left(1 - \sum_{u=L}^{s} \beta_{qT}(u)\right)\right)\right].$$

Now a method to detect the steady state behavior is explained. The method is such that an error introduced due to the steady state detection still has an upper bound less than the specified error tolerance ϵ. Let,

$\pi^* =$ the actual steady-state state occupancy probability vector,

$\pi(S) =$ the detected steady-state state occupancy probability vector, and

$\delta(S) = \parallel \pi^* - \pi(S) \parallel =$ the norm of $\pi^* - \pi(S)$.

If P_{i-} is the ith row of the P matrix and P^s_{i-} is the ith row obtained using the steady state detection, then as shown in [28],

$$\parallel P_{i-} - P^s_{i-} \parallel \leq \frac{\epsilon}{2} + 2\delta(S).$$

By selecting $\delta(S) \leq \epsilon/4$, an error bound of ϵ can be achieved even if the steady state detection is employed. As suggested in [28], at the best, it can be ensured that the norm of the difference between the successive iterations is within the bound $\epsilon/4$.

However, as suggested in [21], if the rate of convergence is too slow, the change in the elements of the vector π between the successive iterations might be smaller than the required error tolerance. In this situation, the steady-state might be incorrectly detected. To avoid this problem two vectors m iterations apart are compared. Based upon the suggestion in [21], m is chosen as follows: $m = 5$ when the number of iterations is less than 100, $m = 10$ when it is between 100 and 1000 and $m = 20$ when it is greater than 1000. Moreover, comparison is done at every mth iteration which helps to minimize the overhead of comparison in the case when the system does not reach steady state. If the rate of convergence is very slow, it is still possible that the steady state might be incorrectly detected.

In the implementation, an option is provided so that the steady state detection can be used. The next chapter presents an application of this implementation to an ATM switch design.

CHAPTER 4

APPLICATION

This chapter shows an application of the solution technique presented in the previous chapter. The application considers switching in an Asynchronous Transfer Mode (ATM) network in which a cell (a basic information packet) service time is deterministic. Basic ATM concepts are reviewed in Section 4.1. The particular switch architecture we considered, called the Low Delay Or Low Loss (LDOLL) switch [1], is described in Section 4.2. The *UltraSAN* model for this switch is presented in Section 4.3, followed by a discussion of the results obtained using the implementation described in the precious chapter. Section 4.5 presents the conclusions reached about the solution technique from the study of LDOLL switch model.

4.1 Asynchronous Transfer Mode

In a broadband integrated services digital network (B-ISDN), with a transmission capacity in hundreds of Mbits/sec, the design is directed towards more flexible assignment of the transmission capacity (bandwidth) in which the bandwidth allocated to a connection is not fixed but depends upon the bandwidth requirement of a source. The ATM network, which can provide capability of flexible bandwidth assignment, consists of switching elements which communicate with each other via high speed networks to obtain virtual circuits and transfer data. As in [36], a switch can be viewed

at different levels of abstraction. The "CONNECTION" level deals with the call acceptance and rejection, the "ACTION" level deals with the bursty characteristics of the traffic and the "TRANSMISSION" level deals with the transmission of the cells. In this study, an ATM switch is considered at the ACTION and TRANSMISSION levels.

Various switch architectures have been proposed in the literature. To achieve a low loss probability for an arriving cell, different buffering schemes are employed. For example, in [13], a thorough comparison of input versus output queueing on an $N \times N$ nonblocking space-division packet switch is presented. In [16], a buffering scheme is presented for the overload control in which after some threshold (L_2), the buffer is reserved only for the higher priority cells until the number of cells in the buffer goes down to another level $L_1 < L_2$. In [17], various priority queueing strategies and buffer allocation protocols for traffic control for the ATM switching system are considered.

The use of ATM helps not only to cater to different "bandwidth on demand" requirements, but also to cater to different performance requirements. In [30], different services which can be offered by the ATM for different performance requirements are discussed. The tradeoff between a low loss and a low delay is discussed along with a "short hold" mode for bursty communications. In the CCITT SG XVIII ISDN experts meeting [3], it was decided that the ATM cell header should have a cell loss priority (CLP) bit. The CLP bit helps to distinguish between two types of cells. Hence two classes of traffic are introduced, one requiring a low loss probability and the other requiring a low delay. Based upon these two types of requirements, and a CLP bit in the ATM cell header, the low delay or low loss (LDOLL) switch was proposed in [1]. Architecture of this switch is reviewed in the next section.

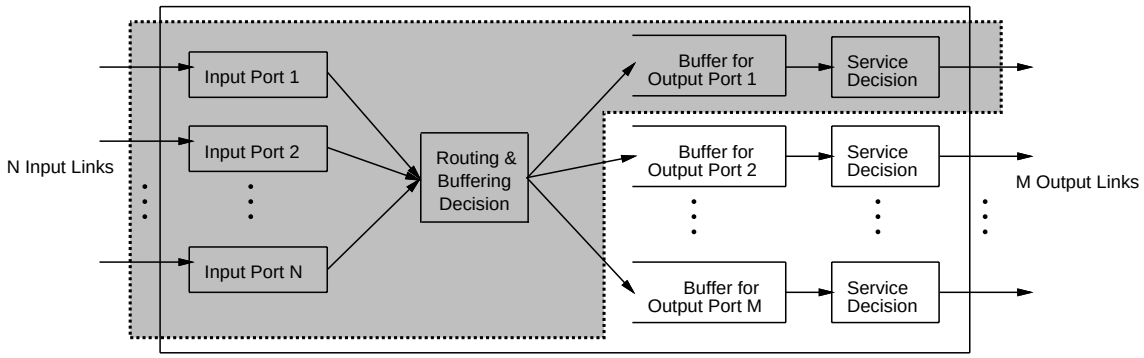

Figure 4.1: Functional Schematic of a Low Delay Or Low Loss Switch

4.2 Low Delay or Low Loss (LDOLL) Switch

Figure 4.1 shows the LDOLL switch. The LDOLL switch has N input ports and M output ports. A buffer, which can hold Q ATM cells, is at the output port of the switch. In a LDOLL switch, two decisions are made which affect the performance of the switch. One decision is made regarding the routing of an arrived cell to a output port and its storage in the appropriate output port buffer when the buffer is full. The other decision is regarding the type of cell to serve from a output port buffer. In this regard, a low loss (LL) cell has a higher priority for storage at the output port buffer and a low delay (LD) cell has a higher priority for service.

More precisely, if an LD cell arrives when the output port buffer is full, it is discarded and considered blocked. If an LL cell arrives when the buffer is full, then it replaces the oldest LD cell in the buffer if at least one LD cell is in the buffer. The LL cell is admitted to buffer and the LD cell is considered replaced. If the buffer is full with LL cells, then the arriving LL cell is lost. Thus an LD cell is lost when it is either blocked or replaced.

Different service policies are possible. Using Markov decision theory and linear

programming, it is shown in [1] that the optimal service policy is a member of the class of "deterministic service policies". The optimal policy reduces the cost (average delays and loss probabilities of interest) associated with it. The term *deterministic service policy* means that for a given state of the switch, the service policy (whether to server an LL cell or an LD cell) is deterministic i.e. not probabilistic nor history dependent.

Optimality is defined as follows. Let, $E(n_{LD})$ = an average number of the LD cells in the buffer and $E(L_{LL})$ = a loss probability for an LL cells. A policy is called *optimal* if for the positive weight coefficients λ and μ, it minimizes $\lambda \cdot E(n_{LD}) + \mu \cdot E(L_{LL})$. No other policy should be able to decrease $E(n_{LD})$ without increasing $E(L_{LL})$ or vice versa. It was conjectured in [1] that among the class of deterministic service policies, the Δ-service policy is the most optimal service policy. In Δ-service policy, as long as the number of LL cells in the buffer are less than Δ, the LD cell at the head of the buffer is served otherwise the LL cell is served.

These buffering and service policies help to achieve a low loss probability for the LL cells as well as a low delay for the LD cells which ultimately get service (the lost or replaced LD cells are not considered in the average delay calculation).

In [1], a small system with $Q = 5$, and $N = 4$ is considered for analytical solution. The Bernoulli process is used as an arrival process with non-bursty traffic. An embedded Markov chain is obtained and solved for steady-state state occupancy probabilities and required performance variables are calculated. A system with $Q = 10$, and $N = 4$ is also considered for a solution by simulation without burstiness in the traffic. The bursty traffic and variable bit rate traffic are considered using simulation in [2] for $Q = 50$.

In this study, an analytic solution of the LDOLL switch with $Q = 16$ and $N = 8$ is considered with bursty traffic of different burst lengths and average channel utilization (burst duty cycle). Large Q and N, and consideration of bursty input traffic results in large (many state) process representations. We use *UltraSAN* to do this and the solution technique presented in the previous chapter to solve the switch model.

The next section explains *UltraSAN* model of the LDOLL switch.

4.3 The Model for LDOLL Switch

Three issues are important in modeling the LDOLL switch:

1. The arrival process,
2. The buffering policy, and
3. The service policy.

The buffering and service policies were explained previously. Several arrival processes are considered in the literature [33]. For example, the Bernoulli process is an arrival process in which cell arrivals on each input ports are independent and identical. To model burstiness, a two state Markov Modulated Poisson Process (MMPP) can be used, which switches the system between ON and OFF periods with the exponentially distributed times T_{on} and T_{off}. During the OFF period there is no input load, while during the ON period the load of 0.99 is considered.

The following assumptions are made in modeling the switch.

1. All the input ports are independent and identical.
2. The arrival process is modeled using MMPP in which, during ON state, arrivals at all the input ports are Bernoulli processes.

Table 4.1: Activity time distributions

Activity	***Distribution***	***Parameter values***
arrival	*instantaneous*	
burst	*exponential*	

	rate	if (MARK(code) == 1) /* ON */
		return(1.0/100.0);
		else /* OFF */
		return(1.0/1000.0);

service	*deterministic*	
	value	1.0 /* a slot time */

3. The arriving cells are uniformly distributed among all the output ports (uniform routing).

4. Cells are buffered at the output ports, and each output port has its own buffer i.e., buffers are not shared.

5. Within the different classes of cells (LD or LL), each cell is severed in the first in first out (FIFO) order.

4.3.1 Model Description

The SAN model of a LDOLL is shown in Figure 4.2. The MMPP is modeled by activity *burst*, output gate *og1*, and place *code*, which changes state between the ON and OFF periods. During the ON period, *code* contains one token and during the OFF period it contains two tokens. The activity time of *burst* depends upon the marking of *code*, as shown in Table 4.1. During the OFF period, all arriving cells are null cells which are not processed by the switch. Only during an ON period, non-null cells arrive (with probability 0.99) and are processed as explained below.

The behavior of the model during one slot time (time to serve a cell) is as follows. Activity *arrival* models the arrival process. At the beginning of the slot time, place

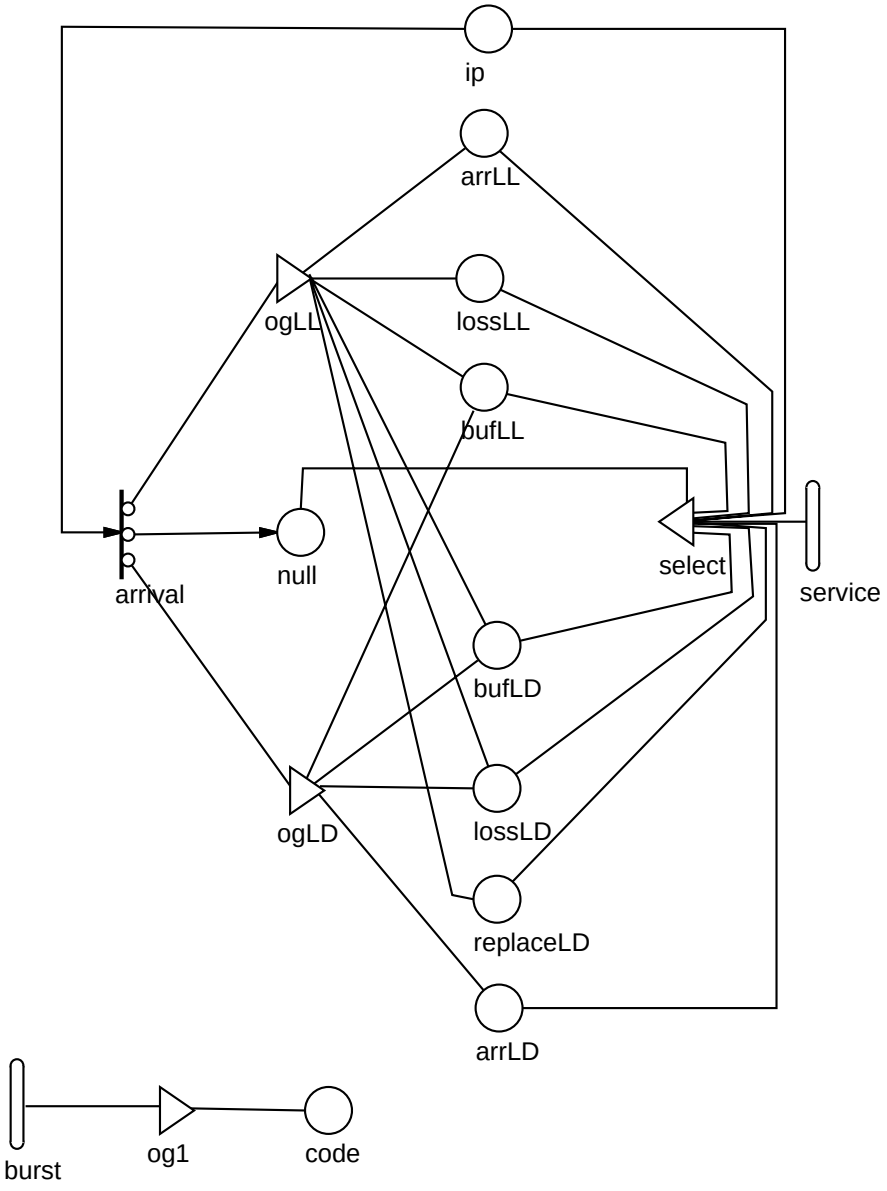

Figure 4.2: SAN Model for LDOLL Switch with Bursty Traffic

ip contains a number of tokens equal to the number of input ports (N). Arrivals for all the input ports are independent and identically distributed. *arrival* models the arrival of cells for all input ports. Let p be the probability that a non-null (LD or LL) cell arrives at the input port, and r be the probability that the arrived non-null cell is the LD cell. *arrival* has three case probabilities. The top case is the probability that an LL cell arrives, the middle case is the probability that a null cell arrives and the bottom case is the probability that an LD cell arrives. Since all the input ports are assumed to be identical, probabilities r and p are the same for all the input ports. Since the traffic is assumed to be uniformly distributed among all the M output ports, the probability that an arriving cell is routed to the output port under consideration, called the tagged output port and highlighted in Figure 4.1, is $\frac{p}{M}$. Hence the three case probabilities of *arrival* are $\frac{(1-r)p}{M}$, $1 - \frac{p}{M}$ and $\frac{rp}{M}$. During the OFF period, these probabilities are 0, 1 and 0. These probabilities are shown in Table 4.2 with $r = 0.5$, $p = 0.99$, and $M = 8$. p is taken to be the input load during the ON period. Thus, *arrival* transfers cells from the input ports to the buffer of the tagged output port. The buffer is modeled by places *bufLD* and *bufLL*.

If an arriving cell is an LD cell, then output gate *ogLD* increments the marking of place *arrLD* by one, which represents the number of arrived LD cells during a slot time. If the buffer is not full (markings of *bufLD* + *bufLL* is less than Q), the marking of *bufLD* is incremented by one (i.e., the LD cell is admitted to the buffer). Otherwise the LD cell is lost and markings of place *lossLD*, which models the blocked LD cells during a slot time, is incremented by one. The function of *ogLD* is shown in Table 4.3.1.

Table 4.2: Case Probabilities for Activities

Activity	***Case***	***Probability***

arrival	1	if (MARK(code) == 1)
		return(0.5*0.99/8.0); /* $\frac{(1-r)p}{M}$ */
		else
		return(ZERO);

	2	if (MARK(code) == 1)
		return(1.0 − 0.99/8.0); /* $1 - \frac{p}{M}$ */
		else
		return(1.0);

	3	if (MARK(code) == 1)
		return(0.5*0.99/8.0); /* $\frac{rp}{M}$ */
		else
		return(ZERO);

If an arriving cell is an LL cell, then output gate *ogLL* increments the marking of place *arrLL*, which represents the number of arrived LL cells during a slot time. If the buffer is not full, the marking of *bufLL* is incremented by one. If the buffer is full and if it has an LD cell, an LD cell from the buffer is replaced by the arriving LL cell. The marking of *bufLL* and place *replaceLD*, which models the replaced LD cells during a slot time, are incremented by one, and the the marking of *bufLD* is decremented by one. If the buffer is full with LL cells then the arriving LL cell is lost. The marking of the place *lossLL*, which models the blocked LL cells during a slot time, is incremented by one. The function of *ogLL* is shown in Table 4.3.1.

If an arriving cell is a null cell, the marking in place *null* is incremented by one. Now the cell at next input port is considered. This process is repeated until all the input ports are considered.

When all the input ports have been processed, the switch serves a cell from the output port. All the input ports are processed at the beginning of the slot time and

Table 4.3: Output Gate Functions

Gate	***Function***

og1	if (MARK(code) == 1)
	  MARK(code) = 2;
	else
	  MARK(code) = 1;

ogLD	#define BUFMAX 16
	
	MARK(arrLD) + +;
	
	if ((MARK(bufLL) + MARK(bufLD)) < BUFMAX)
	{
	  MARK(bufLD) + +;
	}
	else
	  MARK(lossLD) + +;

ogLL	#define BUFMAX 16
	
	MARK(arrLL) + +;
	
	if ((MARK(bufLL) + MARK(bufLD)) < BUFMAX)
	{
	  MARK(bufLL) + +;
	}
	else
	{
	  if (MARK(bufLD) > 0)
	  {
	  MARK(bufLD) − −;
	  MARK(bufLL) + +;
	  MARK(replaceLD) + +;
	  }
	  else
	  MARK(lossLL) + +;
	}

the switch takes time equal to a slot time to serve a cell from the buffer. This time is modeled by activity *service*.

The cell selection logic is implemented in input gate *select*, which is shown in Table 4.4. If the number of LL cells in the buffer is less than the threshold Δ (markings of *bufLL* is less than Δ, where Δ can vary from 1 to Q) and at least one LD cell is present in the buffer, an LD cell is served. Otherwise an LL cell is served. In each cell type, the FIFO service order is implicitly assumed. At the end of service interval (activity time of *service*), the cell being served is removed from the buffer by decrementing the marking of *bufLL* or *bufLD* by one. The marking of places *arrLL*, *lossLL*, *arrLD*, *lossLD*, *replaceLD* are cleared to 0. The marking of *ip* is set to N.

The next service interval begins immediately after the current one is completed.

4.3.2 Performance Variables and Solution

Various performance variables defined at the net level in *UltraSAN* are given in Appendix B. The performance variables of interest are the average delay in the switch and the loss probabilities for both types of cells.

The loss probability of a cell is defined as the average of the ratio of the total lost cells and the total arrived cells per slot time. For the LD cells, the loss probability is a sum of the blocking probability and the replacement probability. The blocking probability is defined as the average of the ratio of the total blocked LD cells (due to the full buffer) and the total arrived LD cells per slot time. The replacement probability is defined as the average of the ratio of the total replaced LD cells (by the LL cells when the buffer is full) and the total arrived LD cells per slot time. For the LL cells, the loss probability is the same as its blocking probability (and the

Table 4.4: Input Gate Predicates and Functions

Gate	***Definition***
select	*Predicate* ((MARK(null) + MARK(bufLL) + MARK(bufLD)) > 0)

	Function
	#define DELTA 8
	
	if ((MARK(bufLL) + MARK(bufLD)) > 0)
	{
	if (MARK(bufLL) < DELTA)
	{
	if (MARK(bufLD) > 0)
	MARK(bufLD) − −;
	else
	MARK(bufLL) − −;
	}
	else /* MARK(bufLL) >= DELTA */
	{
	MARK(bufLL) − −;
	}
	}
	
	MARK(ip) = 8;
	
	MARK(null) = 0;
	MARK(lossLL) = 0;
	MARK(lossLD) = 0;
	MARK(replaceLD) = 0;
	MARK(arrLL) = 0;
	MARK(arrLD) = 0;

replacement probability is zero).

For the LD cell, the blocking probability is obtained by the ratio of the marking of *lossLD* and *arrLD* as shown by the performance variable *Prob_LDcell_lost* in Appendix B. The replacement probability is obtained by the ratio of the marking of *replaceLD* and *arrLd* as shown by the performance variable *Prob_LDcell_replaced* in Appendix B. The LD cell loss probability is obtained by adding the blocking probability and the replacement probability as shown by the performance variable *Prob_LDcell_both*. Similarly the loss probability of the LL cell is obtained by the ratio of the marking of *lossLL* and *arrLL* as shown by the performance variable *Prob_LLcell_lost* in Appendix B.

The average delay in the switch is obtained using Little's result. As shown in [1], the average delay can be found as below.

$$E(delay\ of\ LX\ cell) = \frac{E(number\ of\ LX\ cell\ in\ buffer)}{E(LX\ cell\ arrivals\ per\ slot\ time)}$$

Let B is the duty cycle of the burst period (i.e. $B = \frac{T_{on}}{T_{on}+T_{off}}$). Considering N, p and r as defined previously, the average delays can be found as,

$$E(delay\ of\ LL\ cell) = \frac{E(number\ of\ LL\ cell\ in\ buffer)}{N \cdot (1-r) \cdot p \cdot B \cdot (1 - LL\ cell\ blocking\ probability)}$$

$$E(delay\ of\ LD\ cell) \approx \frac{E(number\ of\ LD\ cell\ in\ buffer)}{N \cdot r \cdot p \cdot B \cdot (1 - LD\ cell\ blocking\ probability)}$$

When the output port buffer is full, the oldest LD cell in the buffer is replaced by an arriving LL cell. So the delay experienced by an LD cell that is eventually served or replaced are almost same. While deriving the equation for the average delay of LD

cell, average delays of replaced or served LD cells are assumed to be exactly the same in [1], which gives a good approximate value for the average LD cell delay. Using the performance variables shown in Appendix B, these delays can be calculated as,

$$E(delay\ of\ LL\ cell) = \frac{ave\_num\_LL}{N \cdot (1-r) \cdot p \cdot B \cdot (1 - Prob\_LLcell\_lost)}$$

$$E(delay\ of\ LD\ cell) \approx \frac{ave\_num\_LD}{N \cdot r \cdot p \cdot B \cdot (1 - Prob\_LDcell\_lost)}$$

After specifying the performance variables, the state space was generated using new RBMC procedure. To solve this SAN, the implemented solver, which can solver SAN with deterministic activities, was used and solution for desired performance variables was obtained. The next section discusses the results obtained.

4.4 Results

For a LDOLL switch with $Q = 16$, $N = 8$, and with bursty traffic, a reduced base model of 17,954 states is generated. In all the states, the deterministic activity *service* is enabled. An error tolerance (ϵ) of 10^{-11} was used for calculating Poisson probabilities, and a convergence accuracy (α) of 10^{-8} was used in iterative solver. The obtained results are discussed in the following paragraphs.

Efficiency of Δ-service policy

The Δ-service policy was conjectured to be the most optimal policy [1]. For the Δ-service policy, since the LL cell loss probability can not be reduced without increasing the LD cell delay or vice versa, the plot of the LL cell loss probability vs. the average LD cell delay, for Δ from 1 to Q, should be a convex hull. The results obtained for

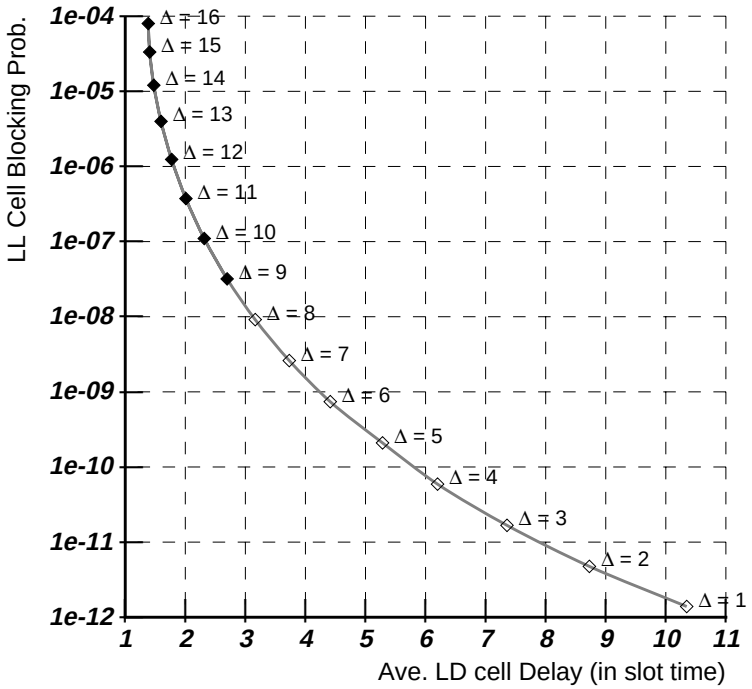

Figure 4.3: LL Cell Blocking Probability vs. Average LD Cell Delay

$Q = 16$, $N = 8$, $p = 0.99$, $r = 0.5$, $T_{on} = 100$ slot times and $T_{off} = 1000$ slot times ($B = \frac{1}{11}$) are shown in Figure 4.3[1] .

Cell Blocking Probability versus Δ

The cell blocking probabilities for the LL and LD cells are plotted vs. Δ in Figure 4.4. The LL cell blocking probability increases with Δ. Since as Δ increases, more LD cells are served than the LL cells. So the probability that the buffer is full the LL cells also increases which increases the LL cell blocking probability. However, the LD cell blocking probability remains constant, which can be explained as follows. At the beginning of the slot time, the available space is given to the arriving LL cells. Any space remaining there after is given to the LD cells. So the probability that the

[1]For this figure and following figures, the results at hollow points are less then the specified convergence accuracy (α).

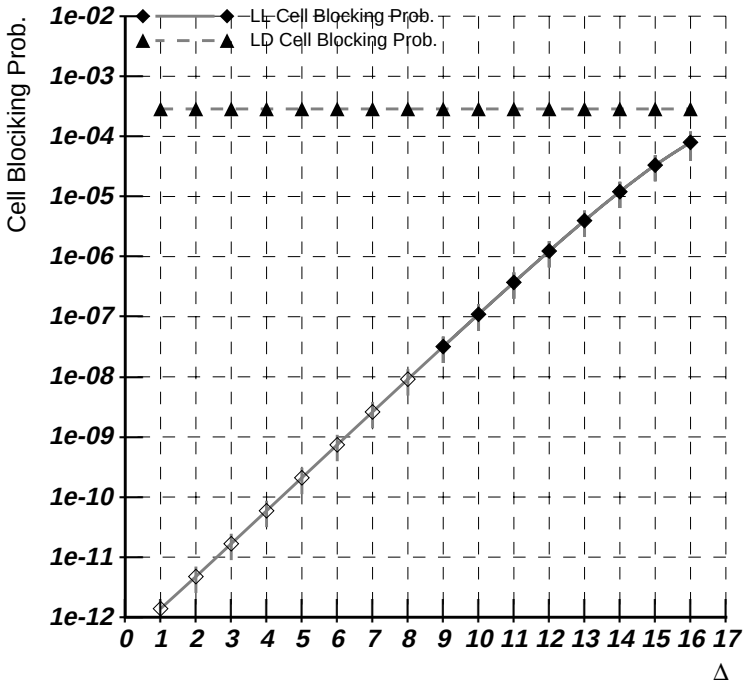

Figure 4.4: Cell Blocking Probability vs. Δ

LD cell finds buffer space does not depend upon which type of cell is served i.e., it is independent of the service policy and depends only on the buffering policy.

Average Cell Delay versus Δ

The average delays for the LL and LD cells are plotted vs. Δ in Figure 4.5. As Δ increases, the average delay for the LL cell increases where as the average delay for the LD cell decreases. The average of both the delays should be equal to the delay if the FIFO service policy would have been employed. Here Δ can be seen as a "delay distributor." If FIFO service policy would have been employed instead of Δ-service policy, the average delay for each type of cell would be around 6 slot times (see point of intersection of both the curves). As Δ changes, this average delay of 6 slot times is redistributed between both the types of cell in such a way that the average of both

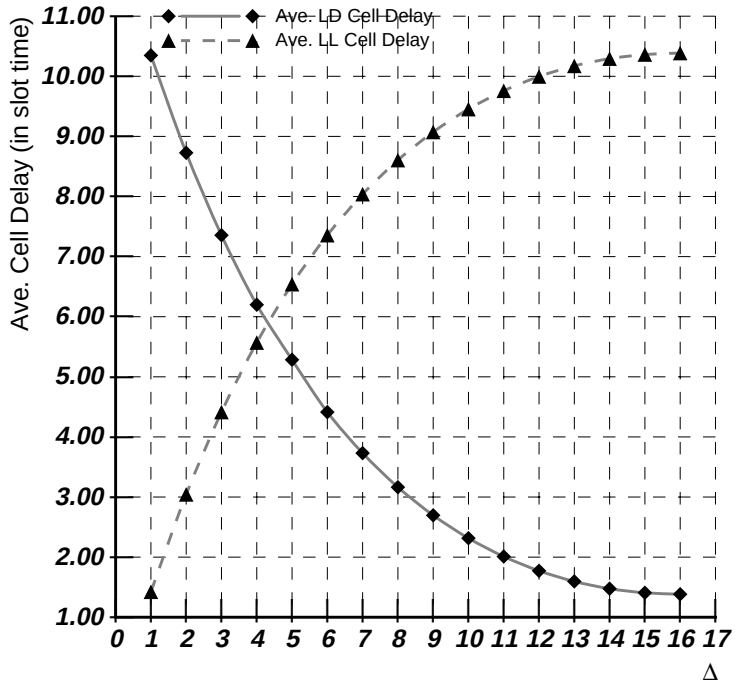

Figure 4.5: Average Cell Delay vs. Δ

the delays is still the same as 6 slot times. For $\Delta < 5$, the average delay for the LL cell is less than the average delay for the LD cell. In this range of Δ, the switch is not fair in the sense that it favors the LL cell for admitting in to the buffer as well as for the service. Moreover, as shown in Figure 4.5, the average delay for the LD cell is the maximum at $\Delta = 1$ which is the same as the maximum average delay for the LL cell at $\Delta = 16$. The same is true for the minimum value of the average delay except that the value of Δ is interchanged.

Effect of Bursty Traffic

The effect of the bursty traffic on the switch is studied by changing T_{on} and the the burst duty cycle $B = \frac{T_{on}}{T_{on}+T_{off}}$. The LL cell and LD cell loss probabilities and the average delays in the switch is plotted vs. T_{on} for various values of $R = \frac{T_{off}}{T_{on}}$ in Figures

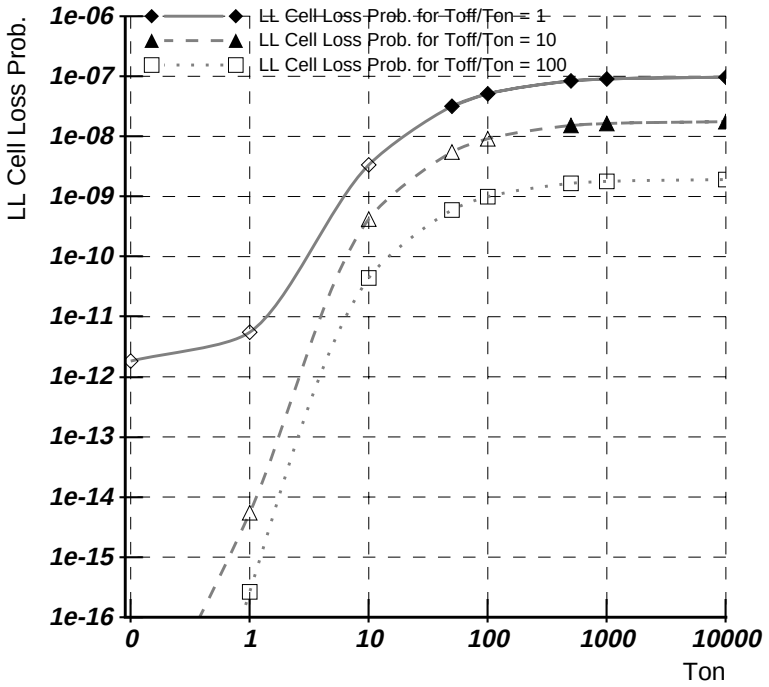

Figure 4.6: LL Cell Loss Probability vs. T_{on}

4.6, 4.7, 4.8 and 4.9 for $Q = 16$, $N = 8$, $\Delta = 8$, $p = 0.99$, and $r = 0.5$.

For each R, as T_{on} increases, the loss probabilities and the average delays for both types of cells also increases. The loss probabilities seems to saturate after $T_{on} = 10000$ slot times for the considered bursty load. The average delays seems to saturate after $T_{on} = 10000$ slot times for the considered load. Moreover after this value of the ON period, the value of the burst duty cycle has no effect on the average delays.

These plots shows the effect of the bursty traffic on the performance of the switch. The highest value of the duty cycle and the highest value of the ON period give the worst case situation in which the loss probabilities and the average delays are the highest. The saturation after $T_{on} = 10000$ suggest that for the given input load, ON period of 10000 slot times (and the corresponding OFF periods) is long enough for the higher utilization of the available buffers.

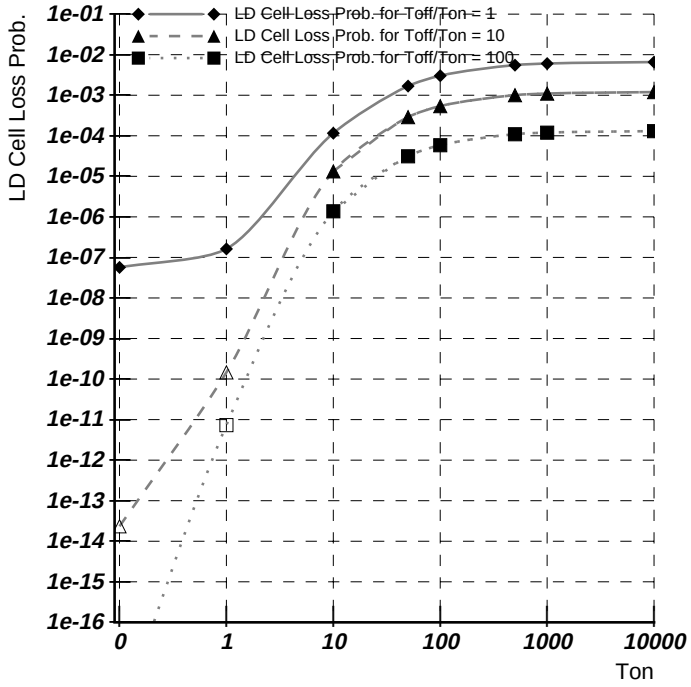

Figure 4.7: LD Cell Loss Probability vs. T_{on}

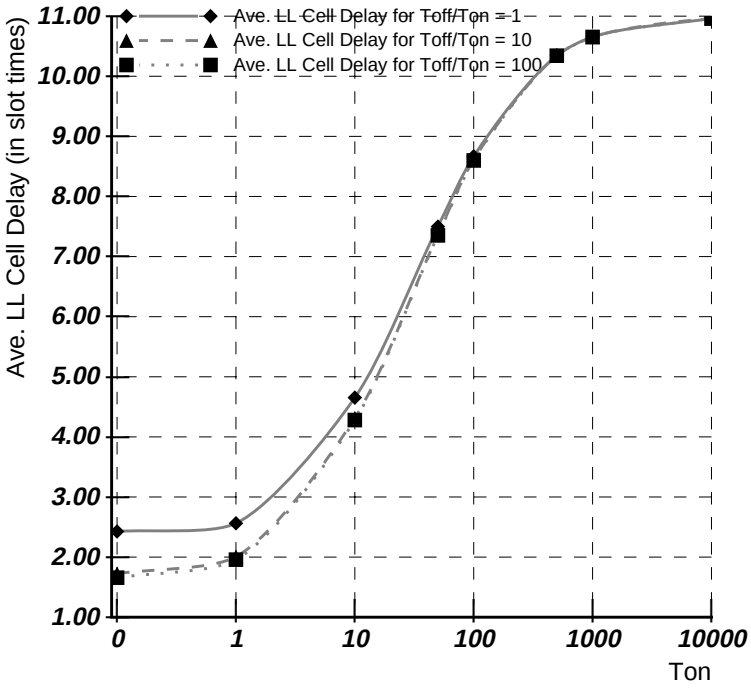

Figure 4.8: Average LL Cell Delay vs. T_{on}

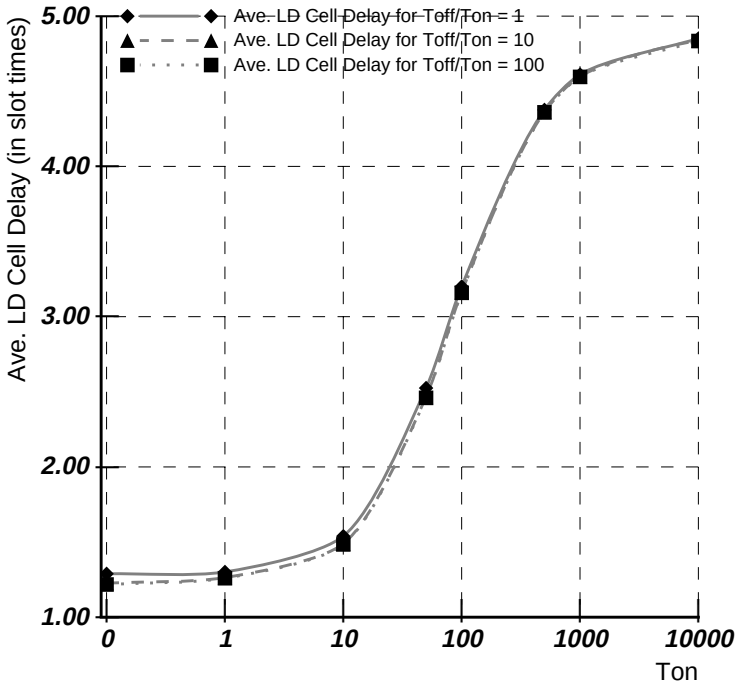

Figure 4.9: Average LD Cell Delay vs. T_{on}

The conclusions about the implemented solver is presented in the next section.

4.5 Conclusions

In addition to the performance results themselves, the study sheds some light on the efficiency of the solution method.

The speed of the solution depends upon the modeled system. The number of uniformization problems that needs to be solved is exactly equal to the the number of stable markings in which a deterministic activity is enabled. For the LDOLL switch, the deterministic activity *service* is enabled in all the stable markings, which is the worst-case scenario. Each uniformization problem takes time depending upon the product of the activity time of the deterministic activity (T) and the maximum diagonal element (q) of the rate matrix, and sparsity of the involved matrices. The

Table 4.5: Efficiency of the Solution Method

$q \cdot T$	Left Truncation Point L	Right Truncation Point R	Solution Time (in seconds)
10	0	39	31506
1	0	12	12179
0.1	0	5	7217
0.02	0	3	5832
0.01	0	3	5862
0.002	0	2	5175
0.001	0	2	5192
0.0001	0	2	5207

greater the product($q \cdot T$), the higher is the right truncation point R. A higher R means more iterations are needed to perform in a single uniformization problem. In a single iteration exactly one vector-matrix multiplication is performed. Each vector-matrix multiplication takes time depending upon the sparsity of the involved vector and matrix. So the greater the product $q \cdot T$, the more vector-matrix multiplications need to solved and the slower the speed.

This fact can be seen in table 4.5, in which product $q \cdot T$ is equal to the maximum diagonal element of the rate matrix (since $T = 1$). The solution time is to solve the constructed reduced base model, using the implemented method on a SparcStation 10/30 with 128 Mbytes of memory. The maximum diagonal element is equal to inverse of T_{on}. Table 4.5 is given for $T_{off} = T_{on}$, but it will not change for any $T_{off} > T_{on}$. For the same right truncation points, solution time varies slightly because of different rates of convergence of the iterative solution method.

CHAPTER 5
CONCLUSIONS AND FUTURE WORK

The conclusions of the efforts made is presented in Section 5.1 and possible areas of further research are suggested in Section 5.2.

5.1 Conclusions

Using the technique of the transient analysis of an embedded Markov chain, the steady-state solution of SANs with deterministic activities can be obtained.

The RBMC technique is augmented so that the reduced base model can be constructed for SANs with deterministic activities. Moreover, the algorithm for a stable and efficient numerical solution technique for SAN models with exponential and deterministic activities is implemented in *UltraSAN*. The usefulness of the implemented solution technique is illustrated by LDOLL switch model.

The speed of the solution depends upon the modeled system. It mainly depends upon the number of stable markings in which a deterministic activity is enabled, the stiffness of the system, and the sparsity of the involved matrices.

A method of steady state detection is included in the solver which can reduce the number of vector-matrix multiplications when R is high but the system can reach the steady-state early.

5.2 Further Work

Significant activities are going on to improve the speed of the solution technique. For example in [20], the graph analysis is done on the generated state space to detect the isomorphisms and special structures. Once a special structure is detected then the closed form solution can be used and the numerical solution can be completely eliminated. If an isomorphism is detected between two components of the state space, the numerical solution is needed for only one of them.

Another direction in the research is towards the transient solution for the systems with the deterministic activities [5].

Another direction in the research is towards the solution for more than one concurrently-enabled deterministic activities. A method of supplementary variables is shown in [7] to solve this problem. In [10], the numerical solution of this method is presented using the Uniformization technique.

Finally, the other direction in the research is towards the general solution technique which should be fast enough to solve any kind of arbitrary system with deterministic activities.

APPENDIX A
DATA STRUCTURES

In this chapter important data structures used in the implementation in *UltraSAN* are presented. The data structures used in the reduced base model construction technique (RBMC) are considered in Section A.1, followed by a section on the data structures used in the iterative solution method. As much as possible, compatibility with existing RBMC and iterative steady state solver is maintained.

A.1 Data Structures for RBMC

The structure *RateList* from the existing RBMC is used without change, but the interpretation of the element *rate* is extended to consider transition probability from one marking to the other. This structure is shown below.

```
/*
 * Each element in the linked list of next states and
 * rates to those states for a given state.
 */
typedef struct rateList {
   int stateNum;
   double rate;    /*
                    * originally rate, Now prob. for DET. ACT. if
                    * 0<rate<=-1 => -ve of prob. to go to stateNum
                    * if rate >= 0 => rate to go to state stateNum
                    */
   struct rateList *next;
} RateList;
```

An array of all the deterministic activities in the system is maintained along with the activity time (an array of structures *DetActs*). For each activity in the array, a linked list of the markings in which this deterministic activity is enabled is maintained (structure *StateInDetAct*). These structures are shown below.

```
/*
 * each DET. ACT. has such structure.
 * It contains time of that activity and
 * pointer to the first state in which this DET. ACT. is enabled.
 */
typedef struct detActs {
   double time;          /* time of the DET. ACT. */
   StateInDetAct *first; /*
                          * pointer to first state in linked list of
                          * states in which the DET. ACT. is enabled
                          */
} DetActs;


/*
 * linked list of states in which a particular DET. ACT. is enabled.
 * Time information for this DET. ACT. is in DetActs
 */
typedef struct stateInDetAct {
   int stateNum;              /*
                               * number of the state in which
                               * DET. ACT. is enabled
                               */
   struct stateInDetAct *next;
} StateInDetAct;
```

For more detail about RBMC technique and its data structures, refer to [32]

A.2 Data Structures for Solver

To maintain the compatibility with the other existing solvers, the data structure for the sparse matrix representation called *struct matrix* as in [37] is used. In this

structure, all the row elements and the corresponding column indices are allocated at the time. While calculating the rows of P and C matrices, the elements of a row are computed one by one. So, the other structures to represent a row of a matrix are employed which are shown below. Since the elements are in a linked list, addition of the elements in a row is easier. A row in this form is converted later in to a row in the form of *struct matrix*.

```
/*
 * this structure points to the first element in the row
 * where all the elements are in a linked list.
 */
typedef struct elemHd {
   long num;      /* total # of elements in the row */
   ElemLL *head;  /* pointer to the first element in linked list */
} ElemHd;


/*
 * in a struct matrix row each has row elements and
 * column indexed which are in a form of an array.
 * This structure can alternatively represents the row.
 * Now each element can be dynamically allocated and added
 * to the existing one by manipulating ``next'' pointer.
 */
typedef struct elemLL {
   double elem;          /* value of an element in a row */
   long index;           /* column index for that element */
   struct elemLL *next;  /* pointer to the next structure */
} ElemLL;
```

APPENDIX B

PERFORMANCE VARIABLES

All the performance variables defined at the net level in *UltraSAN* is presented in the Tables B.1, B.2, B.3, B.4, B.5 and B.6.

Table B.1: Reward variable definitions

ave_num_LL
Rate rewards *Subnet* = node *Predicate:* 1 *Function:* MARK(bufLL)
Impulse rewards *none*

ave_num_LD
Rate rewards *Subnet* = node *Predicate:* 1 *Function:* MARK(bufLD)
Impulse rewards *none*

Table B.2: Reward variable definitions (cont.)

ave_num_loss_LL
Rate rewards *Subnet* = node *Predicate:* 1 *Function:* MARK(lossLL)
Impulse rewards *none*

ave_num_loss_LD
Rate rewards *Subnet* = node *Predicate:* 1 *Function:* MARK(lossLD)
Impulse rewards *none*

ave_num_replace_LD
Rate rewards *Subnet* = node *Predicate:* 1 *Function:* MARK(replaceLD)
Impulse rewards *none*

Table B.3: Reward variable definitions (cont.)

prob_Q_full
Rate rewards *Subnet* = node *Predicate:* ((MARK(bufLL) + MARK(bufLD)) >= 16) *Function:* 1
Impulse rewards *none*

prob_Q_full_with_LL
Rate rewards *Subnet* = node *Predicate:* MARK(bufLL) >= 16 *Function:* 1
Impulse rewards *none*

prob_Q_exceed_delta
Rate rewards *Subnet* = node *Predicate:* ((MARK(bufLL) + MARK(bufLD)) >= 8) *Function:* 1
Impulse rewards *none*

Table B.4: Reward variable definitions (cont.)

prob_lossLL_not0
Rate rewards *Subnet* = node *Predicate:* MARK(lossLL) > 0 *Function:* 1
Impulse rewards *none*

prob_lossLD_not0
Rate rewards *Subnet* = node *Predicate:* MARK(lossLD) > 0 *Function:* 1
Impulse rewards *none*

prob_replaceLD_not0
Rate rewards *Subnet* = node *Predicate:* MARK(replaceLD) > 0 *Function:* 1
Impulse rewards *none*

Table B.5: Reward variable definitions (cont.)

Prob_LLcell_lost
Rate rewards *Subnet* = node *Predicate:* MARK(arrLL) > 0 *Function:* (double)MARK(lossLL) / (double)MARK(arrLL)
Impulse rewards *none*

Prob_LDcell_lost
Rate rewards *Subnet* = node *Predicate:* MARK(arrLD) > 0 *Function:* (double)MARK(lossLD) / (double)MARK(arrLD)
Impulse rewards *none*

Prob_LDcell_replaced
Rate rewards *Subnet* = node *Predicate:* MARK(arrLD) > 0 *Function:* (double)MARK(replaceLD) / (double)MARK(arrLD)
Impulse rewards *none*

Table B.6: Reward variable definitions (cont.)

Prob_LDcell_both
Rate rewards *Subnet* = node *Predicate:* MARK(arrLD) > 0 *Function:* (double)(MARK(lossLD) + MARK(replaceLD)) / (double)MARK(arrLD)
Impulse rewards *none*

ave_num_arrLL
Rate rewards *Subnet* = node *Predicate:* 1 *Function:* MARK(arrLL)
Impulse rewards *none*

ave_num_arrLD
Rate rewards *Subnet* = node *Predicate:* 1 *Function:* MARK(arrLD)
Impulse rewards *none*

REFERENCES

[1] G. A. Awater and F. C. Schoute, "Optimal Queueing Policies for Fast Packet Switching of Mixed Traffic," *Journal on Selected Areas in Communications*. Vol. 9, No. 3, April 1993.

[2] G. A. Awater and F. C. Schoute, "Performance Improvement of Fast Packet Switching by LDOLL Queueing," *IEEE INFOCOM*, Vol. 2, pp. 562-568, 1992.

[3] "CCITT SG XVIII, Draft Recommendation I.361, ATM layer Specification for B-ISDN," *Rep. XVIII-R 23-E*, Geneva, January, 1990.

[4] P. Chen, S. C. Bruell and G. Balbo, "Alternative Methods for Incorporating Non-exponential Distributions into Stochastic Timed Petri Nets," *Proc. 3rd International Workshop on Petri Nets and Performance Models*, Kyoto, Japan, pp. 187-197, 1989.

[5] H. Choi, V. G. Kulkarni and K. S. Trivedi, "Transient Analysis of Deterministic and Stochastic Petri Nets," *Proceedings of the 14th International Conference on Application and Theory of Petri Nets*, Chicago, June, 1993.

[6] J. Couvillion, R. Freire, R. Johnson, W. D. Obal II, M. A. Qureshi, M. Rai, W. H. Sanders and J. Tvedt, "Performability modeling with *UltraSAN*," *IEEE Software*, Vol. 8, no.5, pp. 69-80, September 1991.

[7] D. R. Cox, "The Analysis of Non-Markovian Stochastic Processes by The Inclusion of Supplementary Variables," *Proceedings of Cambridge Philosophical Society (Math. and Phys. Science)*, Vol. 51, pp. 433-441, 1955.

[8] J. B. Dugan, K. S. Trivedi, R. M. Geist, and V. F. Nicola, "Extended stochastic Petri Nets: Applications and Analysis," *Performance 84*, pp. 507-519, Amsterdam, North-Holland 1984.

[9] B. L. Fox and P. W. Glynn, "Computing Poisson Probabilities," *Communications of the ACM*, Vol. 31, No. 4, pp. 440-445, April, 1988.

[10] R. German and C. Lindemann, "Analysis of Stochastic Petri Nets by the Method of Supplementary Variables," *to appear PERFORMANCE*, Rome, Italy, September 1993.

[11] D. Gross and D. R. Miller, "The Randomization Technique as a Modeling Tool and Solution Procedure for Transient Markov Process," *Operations Research*, Vol. 32, No. 2, pp. 343-361, March-April 1984.

[12] A. Jensen, "Markoff Chains as an Aid in the Study of Markoff Processes," Skand, Aktuarietidskrift., 36, pp. 87-91, 1953.

[13] M. J. Karol, M. G. Hluchyj and S. P. Morgan, "Input Versus Output Queueing on a Space Division Packet Switching," *IEEE Transactions on Communications*, Vol. COM-35, No. 12, December 1987.

[14] L. Kleinrock, "Queueing Systems, Volume 1: Theory," John Wiley & Sons, 1975, Theorem 2, page 29.

[15] L. Kleinrock, "Queueing Systems, Volume 1: Theory," John Wiley & Sons, 1975, equations 2.94, 2.95 and 2.104.

[16] S. Li, "Overload Control in a Finite Message Storage Buffer," *IEEE Transactions on Communications*, Vol. 37, No. 12, December 1989.

[17] A. Y. M. Lin and J. A. Silverster, "Priority Queueing Strategies and Buffer Allocation Protocols for Traffic Control at an ATM Integrated Broadband Switching System," *IEEE Journal on Selected Areas in Communications*, Vol. 9, No. 9, December 1991.

[18] C. Lindemann, "An Improved Numerical Algorithm for Calculating Steady-State Solutions of Deterministic and Stochastic Petri Net Models," *Proceedings of the 4th International Workshop of Petri Nets and Performance Models*, pp. 176-185, Melbourne, Australia, December 1991.

[19] C. Lindemann, "DSPNexpress: A Software Package for the Efficient Solution of Deterministic and Stochastic Petri Nets," *Proceedings of 6th International Conference on Modeling Techniques and Tools for Computer Performance Evaluation*, Edinburgh, Great Britain, pp. 15-29, 1992.

[20] C. Lindemann, "Exploiting Isomorphisms and Special Structures in the Analysis of Deterministic and Stochastic Petri Nets," *submitted to Workshop on Petri Nets and Performance Models*, Toulouse, France, 1993.

[21] M. Malhotra, J. K. Muppala and K. S. Trivedi, "Stiffness-Tolerant Methods for Transient Analysis of Stiff Markov Chains," *Technical Report*, DUKE-CCSR-92-001.

[22] M. A. Marsan, G. B. and G. Conte, "A Class of Generalized Stochastic Petri Nets for the Performance Analysis of Multiprocessor Systems," *ACM TOCS* 2(2) pp. 93-122, May 1984.

[23] M. A. Marsan and G. Chiola, "On Petri Nets with Deterministic and Exponentially Distributed Firing Times," *Advances in Petri Nets 1986*, G. Rozenberg (eds.), Lecture Notes in Computer Science 266, pp. 132-145, Springer 1987.

[24] M. A. Marsan and G. Chiola and A. Fumagalli, "Improving The Efficiency of The Analysis of DSPN Models," *Advances in Petri Nets 1989*, G. Rozenberg (eds.), Lecture Notes in Computer Science 424, pp. 30-50, Springer 1990.

[25] J. F. Meyer, A. Movaghar and W. H. Sanders, "Stochastic activity networks: Structure, behavior, and application," *Proceedings of International Workshop on Timed Petri Nets*, pp. 106-115, Torino, Italy, July 1985.

[26] M. K. Molloy, "On the Integration of Delay and Throughput Measures in Distributed Processing Models," PhD Thesis, UCLA, Loss Angeles, CA, 1981.

[27] A. Movaghar and J. F. Meyer, "Performability Modeling with Stochastic Activity Networks," *Proceedings of 1984 Real-Time Systems Symposium*, Austin, TX, December 1984.

[28] J. K. Muppala and K. S. Trivedi, "Numerical Transient Solution of Finite Markovian Queueing Systems," *Queueing and Related Models*, U. N. Bhat and I. V. Basawa (eds.), Oxford University Press, Oxford, UK, 1992, pp. 262-284.

[29] S. Natkin, "Les Reseaux de Petri Stochastiques et leur Application a l'Evaluation des Systemes Informatiques," These de Docteur Ingegneur, CNAM, Paris, France, 1980.

[30] H. Ohnishi, Tadanobu Okada and Kiyohiro Noguchi, "Flow Control Schemes and Delay/Loss Tradeoff in ATM Networks," *IEEE Journal on Selected Areas in Communications*, Vol. 6, No. 9, December 1988.

[31] C. A. Petri, "Kommunication Mit Atomaten, Bonn: Institute fur Instrumentelle Mathematik," Schriften des IIM Nr. 3, 1962.

[32] M. Rai, "Design and Implementation of a Reduced Base Model Construction Technique for Stochastic Activity Networks," MS thesis, The University of Arizona, Tucson, July, 1990.

[33] V. Ramaswami, M. Rumsewicz, W. Willinger and T. Eliazov, "Comparison of Some Traffic Models for ATM Performance Studies," *Proceedings of International*

Teletraffic Congress 13, A. Jensen and V. B. Iversen (eds.), pp. 7-12, 1991.

[34] W. H. Sanders, "Construction and Solution of Performability Models Based on Stochastic Activity Networks," Phd thesis, The University of Michigan, 1988.

[35] W. H. Sanders and J. F. Meyer, "Reduced Base Model Construction Methods for Stochastic Activity Networks," *IEEE Journal on Selected Areas in Communications*, Vol. 9, No. 1, January 1991.

[36] F. C. Schoute, "Simple Decision Rules for Acceptance of Mixed Traffic Streams," *Proceedings of International Teletraffic Congress 12*, Torino, Italy, June 1988.

[37] J. E. Tvedt, "Matrix Representations and Analytical Solution Methods for Stochastic Activity Networks," MS thesis, The University of Arizona, Tucson, Oct., 1990.

[38] V. L. Wallace and R. S. Rosenberg, "Markovian Models and Numerical Analysis of Computer Systems Behavior," *Proceedings of the ACM-IEEE Computer Society Spring Joint Computer Conference*, pp. 141-148, 1966.

[39] D. M. Young, "Iterative Solution of Large Linear Systems," Academic Press, 1971.