# Integrated Reliability and Availability Aanalysis of Networks With Software Failures and Hardware Failures

by

Wei Hou

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
Department of Industrial and Management Systems Engineering
College of Engineering
University of South Florida

Major Professor: O. Geoffrey Okogbaa, Ph.D.
Tapas Das, Ph.D.
A.N. Rao, Ph.D.
Sudeep Sarkar, Ph.D.
Michael Weng, Ph.D.

Date of Approval:
March 17, 2003

Keywords: performance evaluation, distributed systems, system redundancy, end-to-end solution modeling, event tree, application tool

**DEDICATION**

To My Parents

献给我的父母

## ACKNOWLEDGEMENTS

I would like to express my sincere appreciation to my major professor Dr. O. Geoffrey Okogbaa for his academic guidance and financial support to my doctorate research.

I have been indebted to Dr. Tapas Das, Dr. Michael Weng, Dr. Sudeep Sakar, and Dr. A.N. Rao, for the services in my dissertation committee and their precious advice. I am also very thankful to Dr. Rajan Sen for serving as my defense chairperson and Dr. Peter Maurer for his partial service in my committee.

It would be impossible to complete my Ph.D. education, without the support of the Department of Industrial and Management Systems Engineering and its people. I greatly appreciate the help from Dr. William Miller, Dr. Anita Callahan, Ms. Marsha Brett, and Ms. Gloria Hanshaw.

Finally, I am highly grateful of the co-sponsoring of NSF (National Science Foundation) to my dissertation research.

## TABLE OF CONTENTS

## LIST OF TABLES

## LIST OF FIGURES

INTEGRATED RELIABILITY AND AVAILABILITY ANALYSIS OF NETWORKS WITH SOFTWARE FAILURES AND HARDWARE FAILURES

Wei Hou

## ABSTRACT

This dissertation research attempts to explore efficient algorithms and engineering methodologies of analyzing the overall reliability and availability of networks integrated with software failures and hardware failures. Node failures, link failures, and software failures are concurrently and dynamically considered in networks with complex topologies. MORIN (MOdeling Reliability for Integrated Networks) method is proposed and discussed as an approach for analyzing reliability of integrated networks. A Simplified Availability Modeling Tool (SAMOT) is developed and introduced to evaluate and analyze the availability of networks consisting of software and hardware component systems with architectural redundancy. In this dissertation, relevant research efforts in analyzing network reliability and availability are reviewed and discussed, experimental data results of proposed MORIN methodology and SAMOT application are provided, and recommendations for future researches in the network reliability study are summarized as well.

# CHAPTER 1

# INTRODUCTION

## 1.1 Background

The focus of reliability theory studies is the overall performance of a system comprising failure-prone elements. Typically, the components of the system are not perfect with respect to their operation, and their underlying failure structure is assumed to follow certain probabilistic distributions. It is therefore important to characterize the behavior of the system in terms of the stochastic behavior of its components.

The *reliability* of a network is its ability to maintain operational over a period of time $t$. formally, the reliability $R(t)$ of a network is

$$R(t) = Pr \text{ (the network is operational in } [0, t]\}$$

Another measure often used for the analysis of networks is *availability*. The availability of a network is often expressed as *the instantaneous availability* $A(t)$ and/or the *steady-state availability* (i.e., $\lim_{t\to\infty} A(t)$). The $A(t)$ is defined as the probability that a system is operational at time $t$. It allows one or more failures to have occurred during the interval $[0, t]$. If a system is not repairable (e.g., a spaceship), the definition of $A(t)$ is equivalent to $R(t)$. *Dependability* is used as a catch-call phrase for various measures such as reliability, availability etc.

Network reliability is concerned with the interconnectivity of various elements in the form of network, or graph, as exemplified by telecommunication, distribution, and computer networks. For example, the nodes of a computer communication network might represent the physical computers (servers, switches, routers, etc.) and the edges of such a network might represent existing communication links between these nodes. Each node, or edge, or group, or the network can be either operational or failed. Operational in this case means that a specific sender and specific receiver are able to communicate over certain network links, while failure means no complete transmission path is available.

Not only are the reliabilities of individual components of importance, but also the manner in which they are arranged can have a significant effect on the overall dependability performance of the system. For instance, Moore and Shannon [19] configured unreliable components through the use of redundancy to obtain a reliable (high available) system.

The challenge of determining the reliability of a complex system, whose components are subject to failures, has received considerable attention in the engineering, operations research, and statistical literature. Networks have become widely used for modeling complex systems that are subject to component failures.

The earliest use of the stochastic network model was related to analyzing the effects of component or module redundancy in a variety of electronic and mechanical systems [23]. More general networks were analyzed later to determine the effect of blocking in circuit-switched telephone systems. The study of computer communications systems generated

interest in networks with both node and link failures, in both undirected and directed networks, and in measures of reliability more complex than the *2-terminal* system.

In the case of probabilistic networks (where nodes and /or edges fail randomly and independently with known probabilities), a number of measures have been explored. Suppose a network $G$ is directed, with $s$ and $t$ being distinguished nodes of $G$. The 2-*terminal* reliability $R_{st}(G)$ is the probability that there exists at least one path of operating edges in $G$ between $s$ and $t$. The *all-terminal* reliability is the probability that for every pair of nodes there is at least one path between them; equivalently, this is the probability that the graph contains at least one spanning tree. The *k-terminal* reliability of the network is the probability that for $k$ specified target nodes, the graph contains paths between each pair of the $k$ nodes.

The study of network reliability can be categorized into analysis and synthesis. Typical concern about analysis is the computational complexities. It has been shown that network reliability problems with respect to a network with general structure are all *NP*-hard, for *k-terminal*, 2-*terminal*, *all-terminal* in undirected networks, and *all-terminal* in directed networks [4, 17]. Synthesis problem focuses on finding a network topology that satisfies certain deterministic or probabilistic criteria.

Past research in the network reliability field [3, 8-10, 27-30] has focused mainly on networks with perfect nodes and unreliable links. Some of the literatures [2, 5-7, 13-16] have also discussed situations where nodes are subject to failures. However, very few

publications on network reliability field have been found developing the concomitant analysis of both software failures and hardware failures in network nodes [31-32].

## 1.2 Objectives of Research

This dissertation aims to develop efficient approaches to analyze the reliability and availability of networks integrated with node failures, link failures, and software failures. Modeling Reliability for Integrated Networks (MORIN) approach will be proposed and illustrated in Chapter 5 and 7.

Designing handy modeling tools to facilitate the reliability and availability analysis and synthesis is also one of the research objectives to tackle practical network availability problems where integrated systems are subject to hardware failures and software failures, and architectural redundancies are usually deployed at the board level, system level. A Simplified Availability Modeling Tool (SAMOT), which incorporates Markov Analysis and Reliability Block Diagram (RBD) methodologies, is to be developed to address practical network reliability and availability issues, as described in Chapter 6 and 7.

The most common software failure models (such as Jelinski and Moranda model) are to be discussed and applied in computational experiments of the proposed approaches.

## 1.3 Motivation of Research

The study of network reliability is of singular importance due to its clear applicability to computer networks, communication systems, and distribution systems. In certain situations, improving network reliability and availability can be more important than reducing the system cost, especially for mission-critical systems. Reliability analysis can be applied to a variety of practical systems, ranging from large-scale telecommunication system, transportation system, and mechanical system, to integrated circuit boards.

Network reliability is characterized by success of at least one path between two specified nodes. Most of the available researches assume that the nodes of the network are perfectly reliable. However, in a practical communication network or computer network, nodes are also subject to failures with certain probabilities thus under such circumstance reliability evaluation that assumes perfect nodes is not realistic. The evaluation procedure or results are quite complicated and expensive, even for moderately sized networks. So it is quite necessary to develop some simple and efficient approaches.

Major network failures are essentially of three types:

- Node failure due to equipment breakdown or equipment damage resulting from an event such as an accidental fire, flood, or earthquake; as a result, all or some of the communication links terminating on the affected node may fail.
- Link failure due to inadvertent fiber cable cut; despite increased network care and maintenance efforts, the link between one telecommunication office or computer server and the other still fails frequently due to ubiquitous construction activities.

- Software failure that can impact a large portion of the given network, and is, in general, hard to identify and recover from.

Network failures may arise because the routing algorithm is unable to detect a functional route, although one exists. Failures may also arise because the flow control algorithm causes the network to be flooded with traffic, resulting in network failure due to overload. Both events are caused by software control to the network, rather than by topological considerations. In modern information age, software failures, which are shown as traffic congestion, protocol deadlock etc, are very common. Nowadays, software is carrying various types of information and performs more functions, and software reliability is becoming the dominant driver of reliability for complex systems. In a large portion of computer and telecommunication networks, software failures cause more down time than hardware failures do. Software driven outages have been reported to exceed hardware outages by a factor of 10 [11]. Software errors often manifest themselves as network congestion that is quite different from the congestion that arises from hardware failures or traffic overloads. For instance, hardware failures cause congestion by decreasing the number of resources in the network. On the other hand, software errors dramatically decrease the efficiency of network resources used.

During the network operation, failures or errors can also be resulted from changes in the physical state or damage to hardware. Physical changes may be triggered by environmental factors such as fluctuations in temperature or power supply voltage, static discharge. Transient states can be caused by design errors in hardware or software. The

outages of network operation were reported being relatively evenly distributed among hardware, software, maintenance actions, operations, and environment. Table 1.1 depicts the distribution of outages from six different studies [75].

Table 1.1 Probabilities of Operational Outages by Various Causes

| Causes of Outages | AT&T Switching Systems [Toy, 1978] | Bellcore [Ali, 1986] | Japanese Commercial Users | Tandem [Gray, 1987] | Nortel Networks | Mainframe Users |
|---|---|---|---|---|---|---|
| Hardware | 0.20 | 0.26 | 0.25 | 0.19 | 0.19 | 0.45 |
| Software | 0.15 | 0.30 | 0.25 | 0.43 | 0.19 | 0.20 |
| Maintenance | --- | --- | 0.25 | 0.13 | --- | 0.05 |
| Operations | 0.65 | 0.44 | 0.12 | 0.13 | 0.33 | 0.15 |
| Environment | --- | --- | 0.13 | 0.12 | 0.28 | 0.15 |

Note: Dashes indicate that no separate value was reported for that category in the cited study

A lot of research has focused on hardware reliability and software reliability studies. Hardware reliability has reached a nearly mature status and various well-developed hardware reliability techniques have been widely and successfully applied. In the area of software, considerable advances have been made in software reliability modeling, software defect avoidance, software fault-tolerance, and software defect removal (testing). However, this does not solve the reliability problem for network with hardware failures and software failures in a comprehensive way nor does it reveal their inherent relationships. Hence a logic step is to develop appropriate approaches for systems with integrated hardware and software reliability. A number of efforts [78-80] have helped to preliminarily understand the combined hardware-software system reliability.

Analyzing the hardware and software separately by simplifying the system without failures due to interface software might lead to inaccurate estimate of the system reliability [33]. A stochastic process is a mathematical model for description of a probabilistic nature as a function of a parameter that usually has the meaning of time. The set of possible values of the function is the state space of the random variable. The property of a Markov process defines a stochastic process for which the behavior in the future depends only on the present situation, not on the past history. Markov processes with a discrete state space are called Markov chains. Markov chains are accurate, but the state space will explore for large sized networks. Fault tree models can help making accurate analysis, but it is hard to deploy in a real network due to the complex topological relationship between numerous nodes and links.

A comprehensive approach for network reliability analysis has to be developed for practical networks with unreliable components, where link hardware failures, node hardware failures, and node software failures coexist.

## 1.4 Overview of Research

This dissertation consists of eight chapters. Chapter 2 reviews past relevant researches in the area of network reliability, including the application of Petri net (PN) and Colored Petri nets (CPN) in modeling and analyzing the network reliability. Chapter 3 defines and formulates the problem. The most common used approaches for calculating network reliability are introduced in Chapter 4. The proposed approach, namely, MORIN

(MOdeling Reliability for Integrated Networks) is discussed in Chapter 5. Chapter 6 introduces the Simplified Availability Modeling Tool (SAMOT), which incorporates the Markov analysis and RBD methodologies, to model reliability and availability for end-to-end network with system redundancies. Chapter 7 illustrates the MORIN methodology and SAMOT with some examples and numerical experiment results of practical network reliability problems. Chapter 8 summarizes the research and provides recommendations for future researches in the network reliability and availability area.

# CHAPTER 2

# LITERATURE REVIEW

Network reliability and availability researches have made remarkable progress and development in both academic researches and industrial applications. The development of telecommunication systems dates back to the last century with the development of telegraph, telephone, and the transmission, switching and signaling systems supporting them. The forerunner of the internet, the computer communication network ARPAnet was originated in 1969 when the US Department of Defense Advanced Research Projects Agency (ARPA) initiated experiments in resource sharing. Convergence of the two technologies has now occurred with the development of integrated digital networks to support multimedia applications involving voice, data, images and video. The application area covers a vast range of systems embodying traditional telecommunication systems and computer networks, is of utmost importance in the development of new and advanced information systems and services, while maintain or achieve high network availability.

Reliability and availability for integrated networks are becoming vitally important to the global economy. The consequences of failure of the information infrastructure range from minor annoyance to major disruption. It is therefore very important to design and

engineer high available integrated networks according to efficient algorithms, optimized methodologies, rigorous standards, and customer requirements.

Any communication network, computer network, or distributed systems can be modeled as a graph, wherein each node is a switch, computer, or processing entity with its own memory and peripherals, and links are communication lines between nodes. Such a system graph is used in reliability analysis. Moreover, a fault-tree or reliability logic diagram of the system has also been considered. Fault-tree basically translates a physical system into a structured logic diagram and is constructed using the event and logic symbols. In a fault tree, pre-specified causes lead to certain top events of interest. Top events are obtained from a preliminary hazard analysis and usually are undesired system states that could occur as a result of subsystem functional faults.

The reliability block diagram (RBD), on the other hand, shows the functional relationships among resources and indicates which system elements must operate to accomplish the intended function successfully. It should be noted that the RBD is different from the system graph that simply depicts the physical relationship of the system elements. In logic diagrams, if two components must simultaneously function to achieve system success, the blocks representing these corresponding components are shown in series, whereas parallel blocks represent functionally redundant components.

In network analysis, the reliability graph and the system graph could be used interchangeably. Nonetheless, the reliability graph has a probability of operation

associated with each node and with each link. Usually the following basic assumptions are used for the reliability analysis:

- All the elements (nodes and/or links) are always in active mode (no standby or switched redundancy) except stated
- Each element can be represented as a two-terminal device
- The state of each element and of the network is either good (operating) or bad (failed)
- The states of all elements are statistically independent
- The network is free from directed cycles and self-loops, as the success or failure of branches in a directed cycle or self-loop do not alter the terminal reliability

These assumptions are helpful in making the model tractable.

Computer communication networks have evolved in recent years to cope with a massive demand for the information transmission. The interconnection of severs or terminals is achieved by a backbone network. Failures of a LAN (local access network) will affect communications for only a few terminals or end-users, which is not catastrophic. However, backbone failure is usually interpreted as a catastrophic event. Thus most researches in reliability assessment have focused on the synthesis and analysis of reliable backbone network.

## 2.1 Reliability Studies for Networks with Unreliable Links and Perfect Nodes

Most mathematical models for network reliability assume that the network is represented by a graph whose nodes are perfectly reliable and whose edges fail according to some known probabilistic model. There are some traditional approaches to calculate the reliability of networks with unreliable links only [1, 17], as described in Chapter 4.

## 2.2 Reliability Studies for Networks with Unreliable Nodes and Perfect Links

### 2.2.1 Residual Node Connectivity Model

The oldest and most extensively studied model dealing with the case where nodes fail but links are perfectly reliable is the "residual node connectivity model"- first introduced by Frank [43-45]. The network is represented by a simple (no self-loops or parallel links) undirected graph G with node set V and link set E containing 2-element subsets of V. If some sets of nodes fail, these nodes and their incident links are removed from G. The remaining sub-graph is induced by the surviving nodes W, and is denoted by <W>. The links of <W> are those links from E having both endpoints in W. If <W> is connected, the network is operational, and W is an operating state. A reliability function, residual node connectedness reliability, is

$R_n(G, \boldsymbol{p})$ = P(network is operational)

Where $\boldsymbol{p}$ is the vector of $p_v$. If for all nodes, $p_v = p$, then use $R_n(G, \boldsymbol{p})$ or $R_n$. If in additional all nodes operate s-independently of each other, then

$$R_n(G, p) = \sum_{i=1}^{n} S_i p^i (1-p)^{n-i} \qquad (2.1)$$

Where $S_i$ is the number of connected induced sub-graphs of $G$ having exactly $i$ nodes.

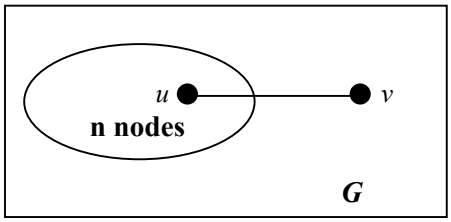
There is an immediate analogy of $R_n$ to the traditional link-failure model where a reliability function for equal link-probabilities is expressed similarly to (2.1) in terms of the number of spanning connected sub-graphs having exactly $i$ links. The coefficients of the link and node reliability functions can also be defined in terms of link cuts or node cuts respectively. It has been determined that calculating $R_n$ is NP-Hard for link failures. However, there are special classes of graphs that admit efficient algorithms for determining $R_n$ [46].

With regard to the synthesis of optimal networks, an important concept is a uniformly optimal network, which has a reliability function that is maximal for all values of $p$ over all networks with the same number of nodes and links. In both the link and node cases, uniformly optimal networks do not always exist [47-51]. Furthermore, some results have been found regarding networks that are optimal for sufficiently small or sufficiently large values of $p$, paralleling results for the link case [47].

Unfortunately the analogy between the link reliability model and the residual node connectedness model is not complete. Indeed the node model has some disturbing properties not shared by the link model.

The model defining $R_n$ assumes that every connected residual graph is acceptable regardless of its size. Figure 2.1 shows an example that is an unusual graph.

The reliability function is not monotone. Making each individual node more reliable can make the network less reliable. Non-monotone behavior is not presented in the link reliability model. Consider any system consisting of a set E of elements and a collection of subsets of E called operating states. If every superset of an operating state is also an operating state, then the system is coherent. Any coherent system has (by definition) a monotone reliability function. The system that defines $R_n$ is not coherent, and is easily verified. Consider *G* in Figure 2.1, the sub-graph *G-u-v* is an operating state. Let node *v*, which was previously failed, be operating. The new resulting induced sub-graph, *G-u*, is disconnected since *v* is isolated. Thus *G-u-v* is an operating state but *G-u* is not.




- If both *u* and *v* fail, the state is operating
- If only *u* fails, the state is failure (not coherent)
- If all nodes except *u* and *v* fail, the state is operating

Figure 2.1 Residual Node Connectedness Reliability Model

The above approach is traditional in the sense that it models network inoperability due to node failure as being caused by node-cuts. This is the direct analog of the link-failure model that uses link-cuts. A few other probabilistic models for studying network vulnerability due to node failure have been introduced. The concept of using the *s*-expected number of node pairs that are connected by a path as a measure of invulnerability was introduced by Amin *et al* [52-53]. This serves as a reasonable approach to the study of graceful and catastrophic degradation of a multiprocessor network. Since this measure is not a probability and thus not reliability, it is difficult to

understand how the results of this approach can be evaluated from the perspective of reliability theory.

An important reliability measure introduced by Fotoh and Colbourn [54-55] contains many results regarding its properties from both synthesis and analysis points of view. It is shown that it is coherent and does not suffer from any of the defects of the residual node connectedness reliability discussed in the foregoing. However, Fotoh and Colbourn described a scenario for their model that a specified set K of nodes (*k*-terminal) are the perfectly reliable hosts or targets that communicate via switching nodes with known probabilities of operating. This important theory, which covers situations like radio frequency (RF) broadcast networks, does not apply to the study of graceful and catastrophic degradation of a multiprocessor network, because in many such networks all nodes are subject to failures.

### 2.2.2 Coherent Model

As the residual node connectedness reliability model has two grievous faults, one might initially consider that an appropriate model could be obtained by a revision of the residual node connectedness reliability model in which only connected sub-graphs of order of at least *k* are defined as operating states. Such a revision corrects the fault that small-connected sub-graphs are considered to be operating states. However, there are two obvious objections to the adoption of this particular revision: a). It is still not coherent in general; b). More importantly, from the standpoint of multiprocessor networks, there is no need to require that every collection of more that *k* nodes induce a connected sub-

graph. The reasonable requirement is to insist that the sub-graph induced by surviving nodes contain a component having at least $k$ nodes.

Boesch *et al* [23] proposed a new coherent model for the problem of obtaining appropriate models for network reliability when the nodes rather than the links are subject to failure. For the application of reliability theory to multiprocessor networks, an operating state is defined as any collection of surviving nodes that induces a sub-graph that contains at least 1 component having $k$ or more nodes. The properties of this model are considered under the additional probabilistic assumption that the nodes fail $s$-independently of each other, all with probability $p$. This is the $k$-node operating component reliability and denoted by, as appropriate $R_{oc}^{(k)}(G, p)$, $R_{oc}^{(k)}(G)$, $R_{oc}^{(k)}$.

The model properties can be observed as,

$$R_{oc}^{(1)}(G, p) = 1 - (1\text{-}p)^n$$

for every $G$ and all $p$, and is trivial. Thus they concentrate on $R_{oc}^{(k)}(G, p)$ for $k \geq 2$.

$$R_{oc}^{(k)}(G, p) = \sum_{i=1}^{n} A_i^{(k)}(G) p^i (1-p)^{n-i}$$

$A_j^{(k)}(G) \equiv$ number of $j$ node induced sub-graphs of $G$ which contain a component having at least $k$ nodes.

$A_j^{(k)}(G) = 0$ for $j < k$,

$$A_j^{(k)}(G) = \binom{n}{j}, \text{ for } j \geq \max(k, n\text{-}k(G) + 1)$$

$A_k^{(k)}(G) = S_k(G)$, (*)

$$A_j^{(k)}(G) \geq S_j(G), \text{ for } k+1 \leq j \leq n$$

The equation (*) shows that the computation of the $k$ node operating component reliability is NP-hard. Indeed if polynomial algorithms exist to calculate $R_{oc}(G)$ for each $1 \leq k \leq n$ and each $0 \leq p \leq 1$, then each $A_k^{(k)}(G)$ can be calculated in polynomial time. However, this means each $S_k(G)$ and therefore $R_n(G)$ can be calculated on polynomial time. But the computation of $R_n(G)$ is NP-hard, hence the calculation of $R_{\text{oc}}^{(k)}(G)$ for all NP-hard.

## 2.3 Reliability Studies for Networks with Unreliable Links and Unreliable Nodes

In a practical telecommunication or computer network, each component of the network is subject to failure. There have been a few approaches proposed to analyze and evaluate the network reliability, considering the node failures [2, 6-10, 13-15].

The methods to evaluate reliability of this type of networks can be classified as explicit or implicit. The explicit has two steps: firstly a symbolic reliability expression presuming perfect nodes is derived, then a special method such as AGM [2] or NPR/T [7] is applied explicitly to the resultant expression to compensate for unreliable nodes. With implicit method, it is unnecessary to apply a special method to account for node failures; the procedure for computing the effect of unreliable nodes is directly embedded into the algorithm and hence it directly computes the reliability expression with unreliable nodes. For instance, ENR/KW [6], TPR/NF [13] and KHR [14] are typical implicit methods to directly obtain the reliability of networks with node failures.

### 2.3.1 AGM Method

To account for node failures, the first and most commonly used method is presented by Aggarwal, Gupta, Misra (AGM). AGM approach has been rigorously proved as a corollary of the general theorem on complex system decomposition. There are some other more efficient algorithms derived from it. However, the computational time of this method increases exponentially with the number of links.

The AGM method considers each link in the network (with link-failure and node-failure probability) as a series combination of a perfect node and the link with modified reliability, as shown in the following figure,

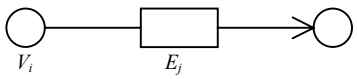



Figure 2.2 Modified Reliability for A Directed Network

In a directed network showed above, the reliability for node $i$ is $\alpha_i$, the reliability for link $j$ is $\beta_j$, the modified reliability for link $j$ is $\beta_j' = \alpha_i\beta_j$.

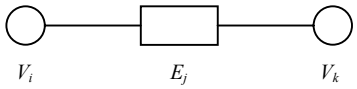



Figure 2.3 Modified Reliability for An Undirected Network

In the interconnecting network, a link can be traversed in both directions. The reliability for node $i$ is $\alpha_i$, the reliability for node $k$ is $\alpha_k$, the reliability for link $j$ is $\beta_j$, the modified reliability for link $j$ is $\beta_j' = \alpha_i\alpha_k\beta_j$.

As a result of the substitution, a particular $\alpha_i$ could appear in a product term more than once. It is necessary to apply an operator to each of these product terms as

$$[\prod_i \alpha_i^{c_i}]^* = [\prod_i \alpha_i]$$

where $c_i$ is the multiplicity of $\alpha_i$. After the traversing, all the nodes can be regarded as perfectly reliable and any algorithms for perfect node networks can be used to derive the reliability.

The AGM method expands each term of the reliability expression derived from perfect nodes and replaces the variables by functions of nodes and link variables. After this substitution, Boolean simplification might be needed. Unfortunately the computing time and cost increase exponentially with the number of links. Furthermore, the use of symbolic calculations rather than direct numerical ones can require prohibitively large storage.

### 2.3.2 NPR/T Method

Torrieri [7] proposed the NPR/T method for calculation of Node-Pair Reliability for large networks with unreliable nodes. In general, NPR/T is much simpler, more direct, and more rigorously derived than AGM, and can compute the same algorithms as AGM. With NPR/T, a set of definite concise formulas is used to capture the relationships between a node and its associated directed links. Therefore the cost of this method rises linearly with the number of links.

For undirected networks, NPR/T should transform the original undirected network into an equivalent directed network wherein each undirected link is replaced with two directed links in anti-parallel; however, such transformation generates s-dependent events in the reliability computation formula and hence, can yield incorrect results for some undirected cases.

### 2.3.3 ENR/KW Method

Based on the concept of network partition, Ke and Wang [6] explored some simple efficient techniques to handle the unreliable nodes, for directly computing the network reliability instead of using any compensating method. The basic idea of ENR/KW is to partition the network directly into a set of smaller disjoint subnetworks by only considering link elements as if all nodes are perfect. Each disjoint subnetwork is generated by maintaining a specific directed graph structure to consider the effect of imperfect nodes. Therefore, the reliability expression for imperfect nodes can be obtained directly from the disjoint subnetwork and the specific directed graph.

## 2.4 Software Models

### 2.4.1 Software Reliability

An important quality attribute of a network is the degrees to which it can be relied on perform its intended function. Until 1960's, attention was almost solely on the hardware related research. In the early 1970's software started becoming a matter of concern,

primarily due to a continuing increase in the cost of software relative to hardware, in both development and the operation phases of the system.

Since software is produced by human beings in a large extent, the finished product is often imperfect in the sense that a discrepancy exists between what the software can do versus that the user or the environment wants it to do. The computing environment refers to the physical machine, operating system, compiler and translator utilities, etc. These discrepancies are called software faults. Basically, software faults can be attributed to ignorance of the user requirements, to ignorance of rules of the computing environment, to poor communication of software requirements between the user and the programmer, or poor documentation of the software by the programmer. Even if we know that software contains faults, we generally do not know their exact identity.

There are two approaches to indicate the existence of software faults: program proving and program testing. Program proving is formal and mathematical while program testing is more practical and heuristic. The approach taken in program proving is to construct a finite sequence of logical statements ending in the statement, usually the output specification statement, to be proved. Each of the logical statements is an axiom or is a statement derived from earlier statements by the application of an inference rule. Program proving by using inference rules is known as the inductive assertion method [56]. Other work on program proving is on the symbolic execution method that is the basis of some automatic program verifiers. Despite the formalism and mathematical exactness, program

proving is still imperfect tool for verifying program correctness. It is showed several programs that were proved to be correct but still contained faults [57].
However the faults were due to failures in defining what exactly to prove and were not failures of the mechanics of the proof itself.

Program testing is the symbolic or physical execution of a set of test cases with the intent of exposing embedded faults in the program. A given testing strategy may be good for exposing certain kinds of faults but not for all possible kinds of faults in a program. An advantage of testing is that it can provide useful information about a program's actual behavior in its intended computing environment, while proving is limited to conclusions about the program's behavior in a postulated environment.

In practice neither proving nor testing can guarantee complete confidence in the correctness of a program. Each has its advantages and limitations and should not be viewed as completing tools. Thus a metric is needed to reflect the degree of program correctness and plan and control additional resources needed for enhancing software quality. One such quantifiable metric of quality is called software reliability. A commonly used approach for measuring software reliability is via an analytical model whose parameters are generally estimated form available measures are then computed from the fitted model.

### 2.4.2 Software Reliability Models

A number of analytical models have been proposed to address the problem of software reliability measurement. These approaches are based mainly on the failure history of software and can be classified according to the nature of the failure process.

#### 2.4.2.1 Time Between Failures Models

This is one of the earliest classes of models proposed for software reliability assessment. When the interest is in modeling times between failures, it is expected that the successive failure times will get longer as faults are removed from the software system.

A number of models have been proposed to describe such failures. The most common approach is to denote the time between the ($i$-1)st and the $i$th failures with a random variable $T_i$. Basically the models assume that $T_i$ follows a known distribution whose parameters depend on the number of faults remaining in the system after the ($i$-1)st failure. The assumed distribution is supposed to reflect the improvement in software quality as faults are detected and removed from the system. Another approach is to treat the failure times as realizations of a stochastic process and use an appropriate time-series model to describe the underlying failure process. The key models in this class are described below.

- Jelinski and Moranda (JM) De-Eutrophication Model

This is one of the earliest and probably the most commonly used model for assessing software reliability. It assumes that there are N software faults at the start of testing, each

is independent of each other and is equally likely to cause a failure during testing. A detected fault is removed with certainty in a negligible time and no new faults are introduced during the debugging process. The software failure rate, or the hazard function, at any time is assumed to be proportional to the current fault content of the program, which is,

$$Z(t_i) = \Phi[N - (i - 1)]$$

Where $\Phi$ is a proportionality constant. This hazard function is constant between failures but decreases in steps of size $\Phi$ following the removal of each fault. A typical plot of the hazard function for $N = 100$ and $\Phi = 0.02$ is shown in Figure 2.4.

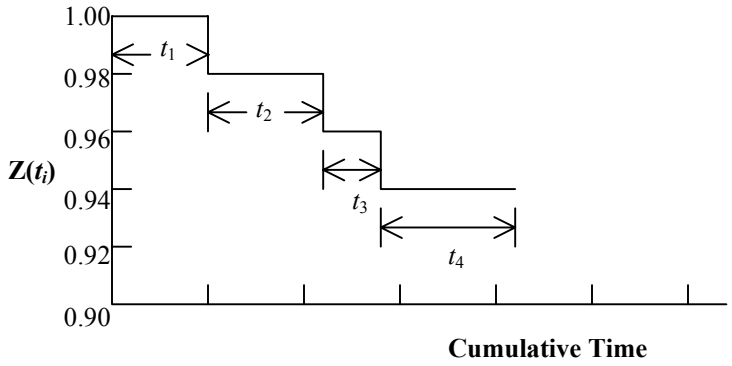



Figure 2.4 A Typical Plot of $Z(t_i)$ for the JM Model ($N = 100$, $\Phi = 0.02$)

A variation of the above model was proposed by Moranda [58] to describe those testing situations where faults are not removed until the occurrence of a fatal one at which time the accumulated group of faults is removed. In such a situation, the hazard function after a restart can be assumed to be a fraction of the rate that attained when the system crashed. For this model, called the geometric de-eutrophication model, the hazard function during the *i*th testing interval is given by

$$Z(t_i) = Dk^{i-1}$$

Where D is the fault detection rate during the first interval and $k$ is a constant ($0 < k < 1$).

- Schick and Wolverton (SW) Model

This model is based on the same assumptions as the JM model that except the hazard function is assumed to be proportional to the current fault content of the program as well as to the time elapsed since the last failure. The hazard function is given by

$$Z(t_i) = \Phi\{[N - (i - 1)]\}t_i$$

The above hazard rate is linear with time within each failure interval, returns to zero at the occurrence of a failure and increases linearly again but at a reduced slope, the decrease in slope being proportional to $\Phi$. A typical behavior of $Z(t_i)$ for N = 150 and $\Phi = 0.02$ is shown in follow Figure 2.5.

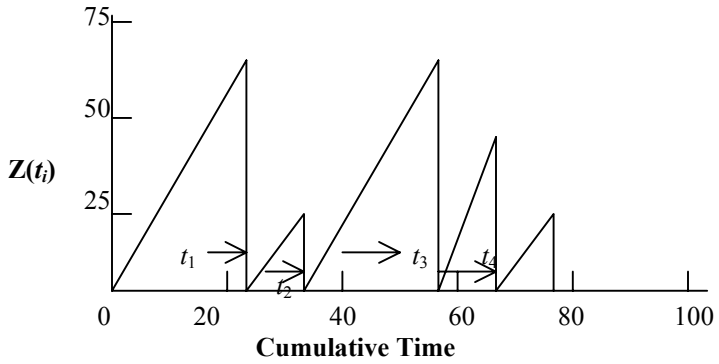



Figure 2.5 A Typical Plot of $Z(t_i)$ for the SW Model ($N = 150$, $\Phi = 0.02$)

A modification of the above model was proposed by Schick and Wolverton [59] whereby the hazard function is assumed to be parabolic in test time and is given by

$$Z(t_i) = \Phi[N - (i - 1)](-at_i^2 + b\,t_i + c)$$

Where $a$, $b$, $c$ are constants and the other quantities are as defined as above. This function consists of two components. The first is basically the hazard function of the JM model and the superimposition of the second term indicates that the likelihood of a failure occurring increases rapidly as the test time accumulates within a testing interval. At failure times ($t_i = 0$), the hazard function is proportional to that of the JM model.

- Goel and Okumoto Imperfect Debugging Model

The above models assume that the faults are removed with certainty when detected. However that is not always true. Goel and Okumoto [60-61] proposed an imperfect debugging model which is basically an extension of the JM model. In this model, the number of faults in the system at time $t$-X($t$) is treated as a Markov process whose transition probabilities are governed by the probability of imperfect debugging. Times between the transition of $X(t)$ are taken to be exponentially distributed with rates dependent on the current fault content of the system. The hazard function during the interval between the ($i$-1)st and the $i$th failures is given by

$$Z(t_i) = [N - p(i\text{-}1)]\lambda$$

Where $N$ is the initial fault content of the system, $p$ is the probability of imperfect debugging, and $\lambda$ is the failure rate per fault.

- Littlewood-Verrall Bayesian Model

Littlewood and Verall [62-63] took a different approach to the development of a model for times between failures. They argued that software reliability should NOT be specified in terms of number of errors in the program. Specifically they assumed the times between failures follows an exponential distribution but the parameter of this distribution is treated as a random variable with a gamma distribution, which is:

$$f(t_i \mid \lambda_i) = \lambda_i e^{-\lambda_i t_i} \quad \text{and} \quad f(\lambda_i \mid \alpha, \psi(i)) = \frac{[\psi(i)]^{\alpha} \lambda_i^{\alpha-1} e^{-\psi(i)\lambda_i}}{\Gamma\alpha}$$

where $\psi(i)$ describes the quality of the programmer and the difficulty of the programming task. It is claimed that the failure phenomena in different environments can be explained by this model by taking different forms for the parameter $\psi(i)$.

2.4.2.2 Failure Count Models

This class of models is concerned with modeling the number of failures seen or faults detected in given testing intervals. As faults are removed from the system, it is expected that the observed number of failures per unit time will decrease. If this is so, then the graph of the cumulative number of failures versus time will eventually level off. The time interval may be fixed a priori and the observed number of failures in each interval is treated as a random variable.

Several models have been suggested to describe such failure phenomena. The basic idea behind most of these models is that of a Poisson distribution whose parameter takes on different forms for different models. It should be noted that Poisson distribution has been

found to be an excellent model in many fields of application where interest is in the number of occurrences.

- Goel-Okumoto Nonhomogeneous Poisson Process Model

Goel and Okumoto [64] assumed that a software system is subject to failures at random times caused by faults present in the system. Letting $N(t)$ be the cumulative number of failures observed by time $t$, they proposed that $N(t)$ can be modeled as a nonhomogeneous Poisson process, *i.e.*, as a Poisson process with a time dependent failure rate. Based on their study of actual failure data from many systems, they proposed the model as

$$P\{N(t) = y\} = \frac{(m(t))^y}{y!} e^{-m(t)} \qquad y = 0, 1, 2, \ldots$$

where $m(t) = a(1 - e^{-bt})$ and $\lambda(t) \equiv m'(t) = abe^{-bt}$

$m(t)$ is the expected number of failures observed by time $t$ and the failure rate. $a$ is the expected number of failures to be observed eventually and $b$ is the fault detection rate per fault. This is a fundamental departure from the other models which treat the number of faults to be a fixed unknown constant.

- Goel Generalized Nonhomogeneous Poisson Process Model

Most of the times between failures and failure count models assume that a software system exhibits a decreasing failure rate pattern during testing. In other words, they assume that software quality continues to improve as testing progresses. In practice, it has been observed that in many testing situations, the failure rate first increases and then

decreases. In order to model this increasing/decreasing failure rate process, Goel [65-66] proposed the following generalization of the Goel-Okumoto NHPP model.

$$P\{N(t) = y\} = \frac{(m(t))^y}{y!} e^{-m(t)} \qquad y = 0, 1, 2, \ldots$$

$$m(t) = a(1 - e^{-bt^c})$$

where $a$ is expected number of faults to be eventually detected, and $b$ and $c$ are constants that reflect the quality of testing. The failure rate for the model is given by

$$\lambda(t) \equiv m' = abce^{-bt^c} t^{c-1}$$

- Musa Execution Time Model

In this model Musa [67] makes assumptions that are similar to those of JM model except that the process modeled is the number of failures in specified execution time intervals. The hazard function for this model is given by

$$z(\tau) = \Phi f(N - n_c)$$

where $\tau$ is the execution time utilized in executing the program up to the present, $f$ is the linear execution frequency (average instruction execution rate divided by the number of instruction in the program), $\Phi$ is a proportionality constant, which is a fault exposure ratio that relates fault exposure frequency to the linear execution frequency, and $n_c$ is the number of faults corrected during (0, $\tau$).

One of the main features of this model is that it explicitly emphasizes the dependence of the hazard function on execution time. Musa also provides a systematic approach for converting the model so that it can be applicable for calendar time as well.

- Shooman Exponential Model

This model is essentially similar to the JM model. For this model the hazard function is of the following form

$$z(t) = k[\frac{N}{I} - n_c(\tau)]$$

Where $t$ is the operating time of the system measured from its initial activation, $I$ is the total number of instructions in the program, $\tau$ is the debugging time since the start of system integration, $n_c(\tau)$ is the total number of faults corrected during $\tau$, normalized with respect to $I$, and $k$ is a proportionality constant.

- Generalized Poisson Model

This is a variation of the NHPP model of Goel and Okumoto and assumes a mean value function of the following form,

$$m(t_i) = \Phi(N - M_{i-1})\, t_i^{\alpha}$$

where $M_{i-1}$ is the total number of faults removed up to the end of the $(i-1)$st debugging interval, $\Phi$ is a constant of proportionality, and $\alpha$ is a constant used to rescale time $t_i$.

- IBM Binomial and Poisson Models

Brooks and Motley [68] consider the fault detection process during software testing to be a discrete process, following a binomial or a Poisson distribution. The software system is assumed to be developed and tested incrementally. They claim that both models can be applied at the module or the system level.

#### 2.4.2.3 Fault Seeding Models

In fault seeding models, a known number of faults is seeded (planted) in the program. The number of exposed seeded and indigenous faults is counted after testing. Using combinatorics and maximum likelihood estimation, the number of indigenous faults in the program and the reliability of the software can be estimated.

- Mills Seeding Model

The most popular and most basic fault seeding model is Mills' Hypergeometric model [69]. This model requires that a number of known faults are randomly seeded in the program to be tested. The program is then tested for some amount of time. The number of original indigenous faults can be estimated from the number of indigenous and seeded faults uncovered during the test by using the hypergeometric distribution.

Lipow [70] modified this problem by considering probability of finding a fault, of either kind, in any test of the software. Then for statistically independent tests, the probability of finding given numbers of indigenous and seeded faults can be calculated. In another modification, Basin [71] suggested a two stage procedure with the use of two programmers to estimate the number of indigenous faults in the program.

#### 2.4.2.4 Input Domain Based Models

The basic approach in the input domain based models is to generate a set of test cases from an input (operational) distribution. Because of the difficulty in estimating the input distribution, the various models in this group partition the input domain into a set of equivalence classes. An equivalence class is usually associated with a program path. The reliability measure is calculated from the number of failures observed during symbolic or physical execution of the sampled test cases.

- Nelson Model

In this input domain based model [72], the reliability of the software is measured by running the software for a sample of $n$ inputs. The $n$ inputs are randomly chosen from the input domain set $E = (E_i: i = 1, \ldots, N)$ where each $E_i$ is the set of data values needed to make a run. The random sampling of $n$ inputs is done according to a probability distribution $P_i$; the set $(P_i: i = 1, \ldots N)$ is the operational profile or simply the user input distribution. If $n_e$ is the number of inputs that resulted in execution failures, then an unbiased estimate of software reliability $\widehat{R} = 1 - \frac{n_e}{n}$. The test set used during the verification phase may not be representative of the expected operational usage.

- Ramamoorthy and Bastani Model

Ramamoorthy and Bastani [73] concerned the reliability of critical, real-time, process control programs where no failures should be detected during the reliability estimation phase, so that the reliability estimate is 1. Thus the important metric of concern is the

confidence in the reliability estimate. This model provides an estimate of the conditional probability that the program is correct for all possible inputs given that it is correct for a specified set of inputs. The basic assumption is that the outcome of each test case provides at least some stochastic information about the behavior of the program for other points that are close to the test points. A main result of this model is

$$P\{\text{program is correct for all points in } [a,\ a+V] \mid \text{it is correct for test cases having successive distances } x_j, j = 1, \ldots, n\text{-}1\} = e^{-\lambda V} \prod_{j=1}^{n-1} \frac{2}{1+e^{-\lambda x_j}}$$

where λ is a parameter which is deduced from some measure of the complexity of the source code.

Unlike other sampling models, this approach allows any test case selection strategy to be used. Hence, the testing effort can be minimized by choosing test cases which exercise error-prone constructs. However, the model concerning the parameter λ needs to be validated experimentally.

## 2.5 Petri Nets in Reliability Analysis of Integrated Networks

### 2.5.1 Introduction of Petri Nets

Petri nets were originally introduced by C.A. Petri in his seminal PhD thesis in 1964, for the study of the qualitative properties of systems exhibiting concurrency and

synchronization characteristics. Although many other models of concurrent and distributed systems have been developed since then, Petri nets are still a central model for concurrent systems with respect to both the theory and applications. They are often used as a yardstick for other models of concurrency. The performance evaluation of communication systems and flexible manufacturing systems, resource allocation problems in information processing systems, communication protocols, production control and process synchronization can be cited as examples of Petri nets applications. This diversity of application has encouraged the study of Petri net theory and both the theory and the applications of this model have been flourishing [90-96] in last decade.

One of the main attractions of Petri nets is the way in which the basic aspects of concurrent systems are identified both conceptually and mathematically. The ease of conceptual modeling (based also on a natural graphical notation) makes Petri nets the model of choice in many applications. The natural way in which Petri nets allow to formally capture many of the basic notions and issues of concurrent systems contributed greatly to the development of a rich theory of concurrent systems based on Petri nets.

#### 2.5.1.1 Evolution of Petri Net Models

The first nets were called Condition/Event Nets (CE-nets). This net model allows each place to contain at most one token – because the place is considered to represent a Boolean condition, which can be either true or false. In the following years a large number of people contributed to the development of new net models, basic concepts, and

analysis methods. One of the most notable results was the development of Place/ Transition nets (PT-nets). This net model allows a place to contain several tokens.

For theoretical considerations, CE-nets are more tractable than PT-nets, and much of the theoretical work concerning the definition of basic concepts and analysis methods has been performed on CE-nets. A new net model called Elementary Nets (EN-nets) was proposed later. The basic ideas of this net model are very close to those of CE-nets – but EN-nets avoid some of the technical problems that turned out to be presented in the original definition of CE-nets.

PT-nets were used for practical applications. But this net model was often too low-level to cope with the real-world applications in a manageable way, and different researchers started to develop their own extensions of PT-nets – adding concepts such as priority between transitions, time delays, global variables to be tested and updated by transitions, zero testing of places etc. In this way a large number of different net models were defined. However, most of these net models were designed with a single, and often very narrow application area in mind. Although some of the net models could be used to give adequate descriptions of certain systems, most of the net models possessed almost no analytic power. The main reason was the large variety of different net models. So it is a difficult task to translate an analysis method developed for one net model to another. The breakthrough with respect to this problem came when Predicate/Transition Nets (PrT-nets) were presented. PrT-nets were the first kind of high-level nets which were constructed without any particular application area in mind. PrT-nets form a

generalization of PT-nets and CE-nets and can be related to PT-nets and CE-nets in a formal way. This makes it possible to generalize most of the basic concepts and analysis methods that have been developed for these net models.

However, PrT-nets present some technical problems when the analysis methods of place invariants and transition invariants are generalized. It is possible to calculate inviriants for PrT-nets, but the interpretations of the invariants is difficult and must be done with great care to avoid erroneous results. The problem arises because of the variables which appear in the arc expressions of PrT-nets. These variables also appear in the invariants, and to interpret the invariants it is necessary to bind the variables, via a complex set of substitution rules. The first version of Colored Petri Nets ($CPN^1$) was defined to overcome this problem. The main ideas of this net model are directly inspired by PrT-nets, but the relation between a binding element and the token colors involved in the occurrence is now defined by functions and not by expressions as in PrT-nets. This removes the variables, and invariants can be interpreted without problems.

Colored Petri nets (CP-nets) have two different representations. The expression representation use arc expressions and guards, while the function representation use linear functions between multi-sets. Moreover, there are formal translations between the two representations. The expression representation is nearly identical to PrT-nets, while the function representation is nearly identical to CPN. Most of the practical applications of Petri nets use either PrT-nets or CP-nets although several other kinds of high-level nets have been proposed. The main difference between PrT-nets and CP-nets are hidden

inside the methods to calculate and interpret place and transition invariants. So PrT-nets and CP-nets are viewed as two slightly different dialects of the same language due to very little difference between them.

Several other classes of high-level nets include algebraic nets, CP-nets with algebraic specifications, many sorted high-level nets, numerical Petri nets, OBJSA nets, PrE-nets with algebraic specifications, Petri nets with structured tokens and relation nets. All these net classes are quite similar to CP-nets but use different inscription languages. The functional programming language Standard ML has been developed at Edinburgh University and is used for the inscriptions of CP-nets. It is also one of the programming languages used in the implementation of the CPN tools described in section 2.5.3.

"Petri nets" is a generic name for a whole class of models that can be divided into three main layers. The first layer is the most fundamental and is especially well suited for a thorough investigation of foundational issues of concurrent systems. The basic model is that of elementary net systems or EN-nets [110-112]. For modeling real-life systems of nontrivial size, elementary net systems may explode in size and become much too large to be managed effectively. The second layer allows one to collapse the repetitive features of elementary net systems in order to get more compact representations. The basic model here is place/transition systems or PT-nets [113-114]. Finally, the third layer is that of high level nets, where one uses essentially algebra and logic to yield compact nets suitable for real-life applications. Colored Petri nets [103] and predicate/transition nets (PrT-nets) [115] are the best known high-level models.

In the framework of EN systems, a concurrent system is seen as consisting of local states, local transitions (between local states), and the neighborhood relationship between the local transitions and the local states. The global state of a system (its configuration) is simply the collection of all local states that concurrently hold. The extent of change caused by a (local) transition is fixed and is restricted to the neighborhood of the transition; it does not depend on the part of the global state that is outside the neighborhood. This simple and elegant setup lends itself to a nice graphical representation of both the static structure of the system and its dynamic behavior.

The EN system model has resulted from a number of modifications of the basic system model called Condition/Event Systems, or CE-nets. The most significant difference is that CE-nets transitions can also be reserved, recovering in this way the history of the system. An EN system can also be viewed as a special case of a PT-net.

For many practical applications, the execution time and/or stochastic processes need to be considered. This leads to *timed and stochastic Petri nets*.

#### 2.5.1.2 Definitions of Petri Nets

Petri net definitions have a “static” part and a “dynamic” part. The former describes net topology and a momentary marking. The latter describes the movement of tokens in time via a switching (or firing) rule.

A Petri net is a bipartite directed graph. It consists of two types of nodes: *places* (drawn as circles), which can be marked with *tokens* (drawn as bold face dots), and *transitions* (drawn as squares), which are marked by the (random or deterministic) time, D by which they delay the output of tokens. If D = 0, the transition is called *immediate*; otherwise it is called *timed*. The movement of tokens is governed by so-called *firing rule*. If all input places of a transition are marked by at least one token each, then this transition is called *enabled*; and after a delay D ≥ 0 this transition switches or fires, i.e., it removes one token from each of its input places and adds one to each of its output places. See Figure 2.6, where place 3 ($p_3$) is at the same time an input and an output of transition 1, $t_1$.

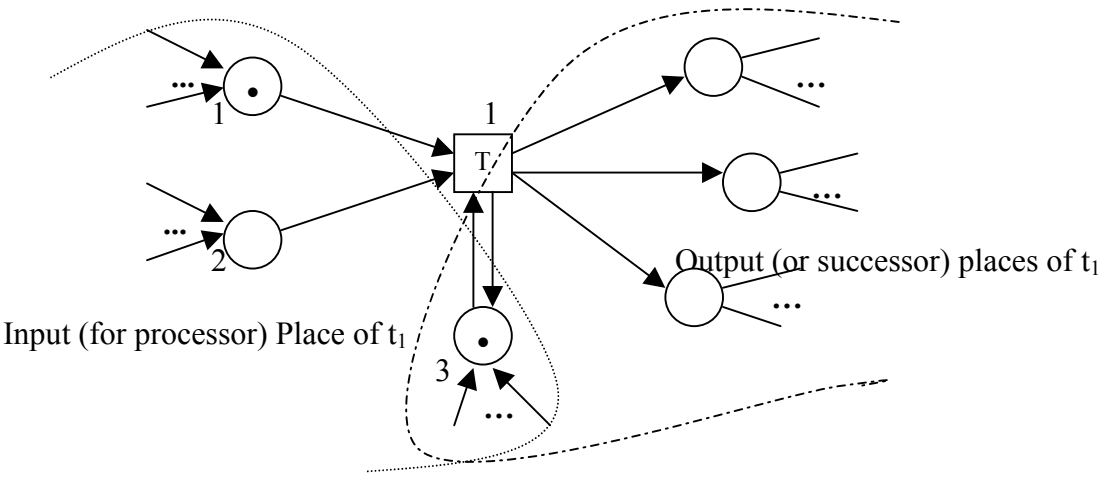



Figure 2.6 Input and Output Places of A Transition

The number of tokens in a Petri net is not necessarily a constant. Tokens move along (or through) edges at infinite speed. Figure 2.7 shows an example of a transition with 3 input places and 2 output places.

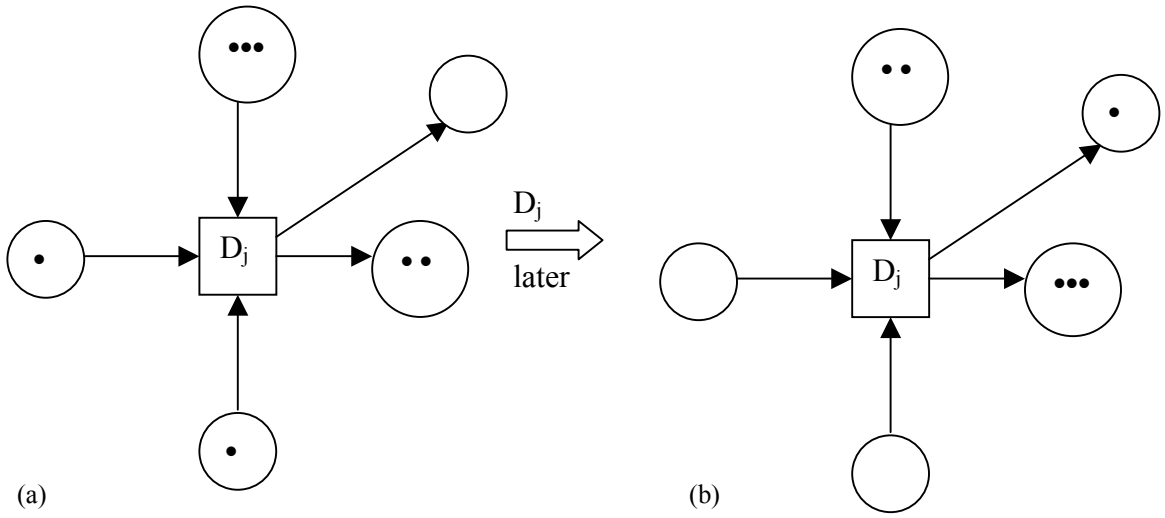



Figure 2.7 The Delayed Switching of A Transition; (a) prior to, (b) after switching

If a PN is initially a multigraph as shown in Figure 2.8, then it is replaced by a graph with weighted edges where the default value is 1. The transition of Figure 2.8 is not enabled, since $p_2$ has only one token but needs at least 2 for firing.

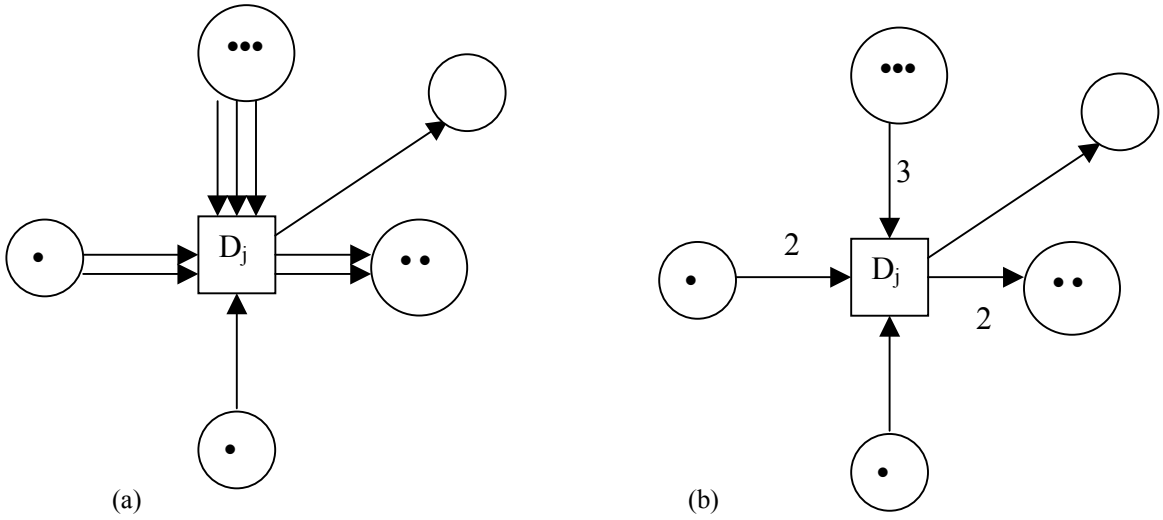



Figure 2.8 Replacing A Multigraph by A Graph With Weighted Edges

#### 2.5.1.3 Timed Petri Nets (TPN)

One of the main attractions of Petri nets is the way in which the basic aspects of concurrent systems are identified both conceptually and mathematically. The ease of conceptual modeling (based also on a natural graphic notation) makes Petri nets the model of choice in many applications.

Petri nets (PN) were originally developed and used for the study of the qualitative properties of systems exhibiting concurrency and synchronization characteristics. The use of PN-based techniques for the quantitative analysis of systems requires the introduction of temporal specifications within the basic, untimed models. This fact leads to several different proposals for the introduction of temporal specifications in PN.

The main alternatives that characterize the different proposals concern

- The PN elements associated with timing (normally either places or transitions, but some also looked into the possibility of defining timed arcs or tokens),
- The firing semantics in the case of timed transitions (either atomic firing or firing in three phases),
- The nature of the temporal specification (either deterministic or probabilistic),
- The conflict resolution policy.

We consider PN models that are augmented with a temporal specification by associating a (possibly null) firing delay with transitions. The transition firing operation is assumed to be atomic, i.e., tokens are removed from input places and put into output places with a

single, indivisible operation, after the transition firing delay has elapsed. The specification of the firing delay of timed transitions is of probabilistic nature, so that either the probability density function (pdf) or the cumulative distribution function (cdf) of the delay associated with a transition needs to be specified. Such functions may be general, or even degenerate, thus allowing the definition of constant (possibly null) delays. We refer to this type of timed Petri nets as Generally Distributed Times Transitions Stochastic Petri Nets (GDTT_SPN).

The class of TPN is however too wide to allow a simple solution of any GDTT_SPN model; so special attention are paid to two special subclasses of GDTT_SPN, that have nice property of permitting a reasonably simple representation metrics:

- Stochastic Petri Nets (SPN), where all transition firing delays are non-null and have negative exponential pdf.
- Generalized SPN (GSPN), where immediate (null-delay) transitions are freely mixed with timed transitions associated with exponentially distributed non-null random firing delays.

A SPN is a GDTT_SPN in which the W function assigns to each transition an exponential pdf. Since the exponential distribution is fully characterized by its mean value (or by its inverse, the rate), and its memory-less characteristics makes inessential.
The definition of a SPN is SPN = (P, T, I, O, H, $M_0$, W)
Where - (P, T, I, O, H, $M_0$) is the underlying PN system, as for GDTT_SPN,

W: T $\rightarrow$ R is a weight function; w(t) is the rate of the exponential distribution associated with transition t. w(t) is also called the firing rate of transition t.

The key factor that limits the applicability of SPN models is the complexity of their analysis. The possibly very large number of reachable markings is by far the most critical reason among many other reasons. Other aspects may however add to the model solution complexity. One of these is due to the presence in one model of activities that take place on a much faster (or slower) time scale than the one relating to the events that play a critical role on the overall performance. This results in systems of linear equations which are difficult to solve with an acceptable degree of accuracy by means of the usual numerical techniques. On the other hand, neglecting the "fast" (or "slow") activities may result in models which are logically incorrect.

GSPN models comprise two types of transitions:

- Timed transitions, which are associated with random, exponentially distributed firing delays, as in SPN, and
- Immediate transitions, firing in zero time with priority over timed transitions.

Furthermore, different priority levels of immediate transitions can be used, and weights are associated with immediate transitions.

### 2.5.2 Colored Petri Nets

A Colored Petri Net (CPN) model of a system describes the states a system can get into, and shows events which can occur and the states which will result if an event occurs for

each state. A CPN state is broken into a number of component states, each component being determined by tokens in a place. Tokens can have arbitrary values determined by their type or color. Each distinct token value can be thought of as a different colored or shaped piece on a board game. The places are like the parts of a game board where you can put pieces. Events are represented by transitions. They are connected to some of the places by arcs next to which are expressions that determine the redistribution of tokens that occurs when the event occurs.

High level Petri nets, such as CPN and SPN have the particular feature of presenting concise and easy to understand graphical models that visualize the interactions between the different communicating and cooperating entities of the system. The applications of high level Petri Nets to the modeling and simulation of communication protocol has increased in recent years [97-103].

CPNs, and especially Hierarchical CPN (HCPN)[103], are the response to the first requirement, as they have means for modeling and specifying very large scale systems, with their colored tokens and hierarchy constructs, folding the system description into very compact forms. While SPNs (with its extensions, GSPNs and Deterministic SPNs) constitute an answer to the second requirement, as they can be useful in modeling complex system with a very high level of abstraction.

#### 2.5.2.1 Advantages of Colored Petri Nets

There are three different reasons to use CPN models. First of all, a CPN model is a description of the modeled system, and it can be used as a specification (of a system which we want to build) or as a presentation (of a system which we want to explain). By creating a model we can investigate a new system before constructing it. This is in particular for networks where design errors may jeopardize reliability or be expensive to maintain. Secondly, the behavior of a CPN model can be analyzed, either by means of simulation (which is equivalent to program execution and program debugging) or by means of more formal analysis methods (which are equivalent to program verification). Finally, the process of creating the description and performing the analysis usually gives the modeler a dramatically improved understanding of the modeled system.

There exist many different modeling languages that it would be very difficult and time consuming to make an explicit comparison with all of them. Instead we can make an implicit comparison by listing twelve of those properties which make CPN a valuable language for the design, specification and analysis of many different types of systems. Most of the advantages of CPN are subjective by nature and cannot be proved in any formal way. Jensen [94] presented the general list of CPN advantages.

- CPNs have a graphical presentation. The graphic form is intuitively appealing. CPN diagrams resemble many of the informal drawings which designers and engineers make while they construct and analyze a system.
- CPNs have a well-defined semantics which unambiguously defines the behavior of each CPN. It is the presence of the semantics which makes it possible to

implement simulators for CPNs, and it is also the semantics which forms the foundation for the formal analysis methods.

- CPNs are very general and can be used to describe a large variety of different systems. The CPN applications range from informal systems (e.g. the description of work processes) to formal systems (e.g. communication protocols), from software systems (e.g. distributed algorithms) to hardware systems (e.g. VLSI chips), finally from systems with a lot of concurrent processes (e.g. flexible manufacturing) to systems with no concurrency (e.g. sequential algorithms).
- CPNs have very few, but powerful, primitives. The definition of CPNs is rather short and it builds upon standard concepts which many system modelers already know from mathematics and programming languages. This means that it is relatively easy to learn to use CPNs. However, the small number of primitives also means that it is much easier to develop strong analysis methods.
- CPNs have an explicit description of both states and actions. This is in contrast to most system description languages which describe either the states or the actions but not both. At some instances it may be convenient to concentrate on the states while at other instances it may be more convenient to concentrate on the actions.
- CPNs have a semantics which builds upon true concurrency, instead of interleaving. The notions of conflict, concurrency and casual dependency can be defined in a very natural and straightforward way. In an interleaving semantics it is impossible to have two actions in the same step, and thus concurrency only means that the actions can occur after each other, in any order.

- CPNs offer hierarchical descriptions. This means that we can construct a large CPN by relating smaller CPNs to each other, in a well-defined way. The hierarchy constructs of CPNs play a role similar to that of subroutines, procedures and modules of programming languages, and it is the existence of hierarchical CPNs which makes it possible to model very large systems in a manageable and modular way.
- CPNs integrate the description of control and synchronization with the description of data manipulation. This means that it can be seen what the environment, enabling conditions and effects of an action are. Many other graphical description languages work with graphs which only describe the environment of an action – while the detailed behavior is specified separately.
- CPNs are stable towards minor changes of the modeled system. This is proved by many practical experiences and it means that small modifications of the modeled system do not completely change the structure of the CPN.
- CPNs offer interactive simulations where the results are presented directly on the CPN diagram. The simulation makes it possible to debug a large model while it is being constructed – analogously to a good programmer debugging the individual parts of a program as he finishes them.
- CPNs have a large number of formal analysis methods by which properties of CPNs can be proved. There are four basic classes of formal analysis methods: construction of occurrence graphs (representing all reachable markings), calculation and interpretations of system invariants (called place and transition invariants), reductions (which shrink the net without changing a certain selected

set of properties) and checking of structural properties (which guarantee certain behavioral properties).

- CPNs have computer tools supporting their drawing, simulation and formal analysis. This makes it possible to handle even large nets without drowning in details and without making trivial calculation errors. The existence of such computer tools is extremely important for the practical use of CPNs.

Many of above listed advantages of CPNs are also valid for other kinds of high-level nets, P/T nets, and other kinds of modeling languages. Thus CPNs must be used together with other kinds of modeling languages to describe different aspects of the system, then the resulting set of descriptions should be considered as complementary, not alternatives.

### 2.5.3 Tools for Petri Nets Applications

There have been a lot of tools for Petri Nets (PN) applications, with the development of Petri Nets theory. The simplest PN tool shows the typical changes of state, sometimes interpretable as the wandering of tokens and the waiting times in between. This is often done in connection with a graphical display of the PN. Some other tools include:

- SHARPE [105]
- Great SPN [106]
- ESP [107]
- Ultra SAN [108]
- SPNP [109]

### 2.5.4 PN_RAIN Approach

A practical network is usually subject to node failures, link failures, and software failures, where node failures and link failures here are viewed as failures on hardware aspect. Each type of failure can occur concurrently, as in Figure 2.9.

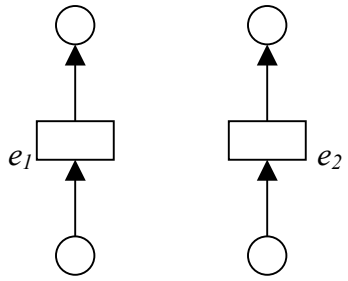



Figure 2.9 Sample Concurrent Events

The failure events $e_1$ and $e_2$ can occur concurrently, in the sense that they both have concession and are independent in not having any pre or post conditions in common. Reflecting to the network under study (refer to Chapter 3), that means node failures, link failures, and software failures can occur concurrently in general, but two failures can not occur at the same time among a node and its incident links.

Taking the networks described in Chapter 3 as the research object, an approach of Petri Nets in Reliability Analysis of Integrated Networks (PN_RAIN) will be introduced.

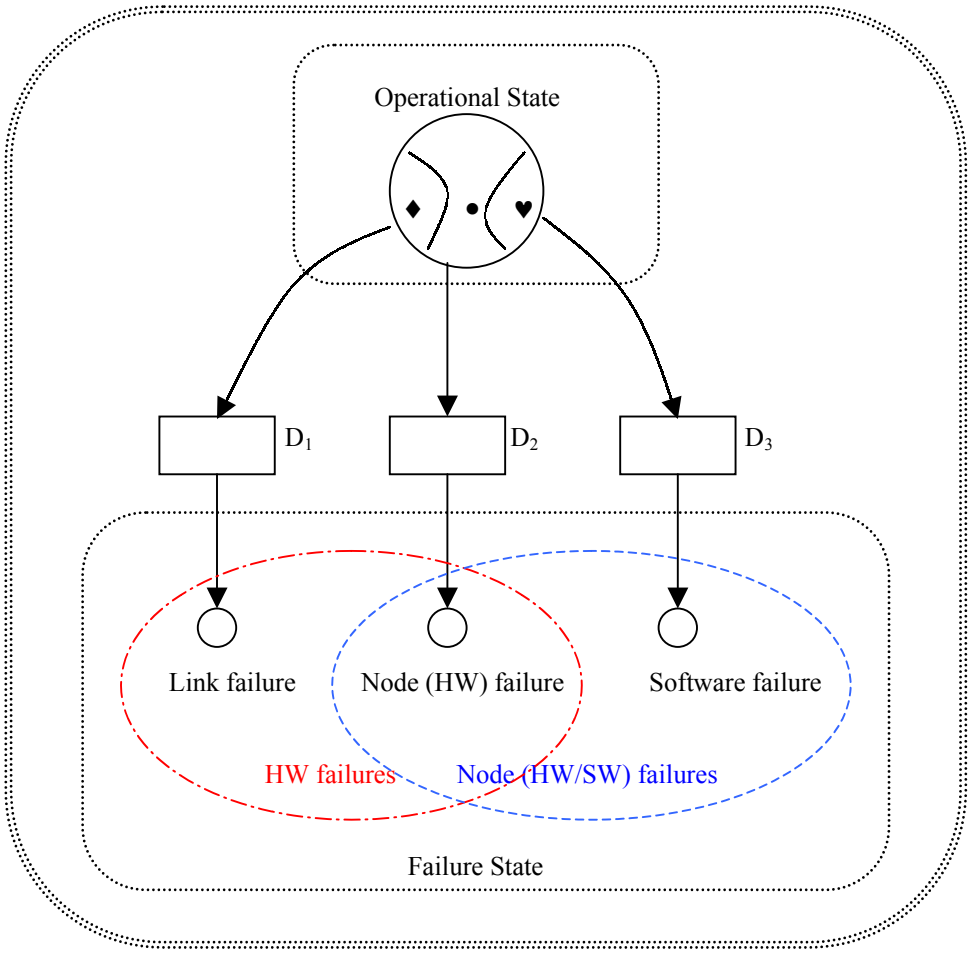



Figure 2.10 States Transition of A Node in An Integrated Network

Generally there are three types of failure processes, initiated by link failures, node failures, and software failures. Link failures represent failures associated with links incident to the node. The three failure processes are independent and concurrent. In Figure 2.10, there are three different colors of tokens representing three types of failures. Each of $D_1$, $D_2$, and $D_3$ represents the firing delay of each type of token correspondingly. In a practical network, each type of firing delay follows the stochastic distribution of link failures, or node (hardware) failure, or software failures. Figure 2.10 represents a node in

an integrated network. There are four nodes in Figure 4.1, thus the node state in Figure 2.10 can replicate four times, as shown in Figure 2.11.

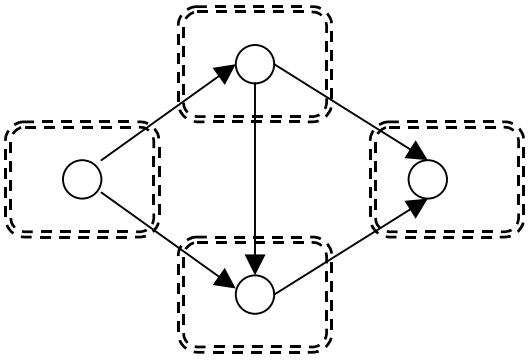


Figure 2.11 A Sample Bridge Network (Figure 4.1) With Node States

#### 2.5.4.1 Construction of PN_RAIN Models

For all modeling languages, it takes a considerable amount of experience to become a good and efficient CPN modeler. The construction of CPN models usually follows:

- Identify some of the most important components of the modeled system.
- Consider the purpose of the model and determine an adequate level of detail.
- Try to find good mnemonic names for objects, processes, states and actions.
- Do not attempt to cover all aspects of the considered system in the first version of the model.
- Choose one of the processes in the modeled system and try to make an isolated net for this process.
- Use the net structure to model control and the net inscriptions to model data manipulations.

- Distinguish between different kinds of tokens.
- Use different kinds of color sets.
- Augment the process net by describing how the process communicates/ interacts with other processes.
- Investigate whether there are classes of similar processes.
- Combine the subnets of the individual process to a large model.

Assume we have two types of processes, N-processes (for node) and L-processes (for link). There are four N-processes and five L-processes in a network depicted by Figure 2.11. A N-process is subject to the node (hardware) failures and software failures. Since the failure of either hardware or software of a node will bring its incident links down, a L-process is subject to failures of its incident nodes and link itself. Obviously node failures, software failures and link failures follow different stochastic distributions, but we assume same type of failure follows the same stochastic distribution in different processes. There is only one token in each place, which means one type of failure can only occur once among the corresponding node and its links. When any failure (by nodes or links) transition is enabled and fired, the state of the system changes.

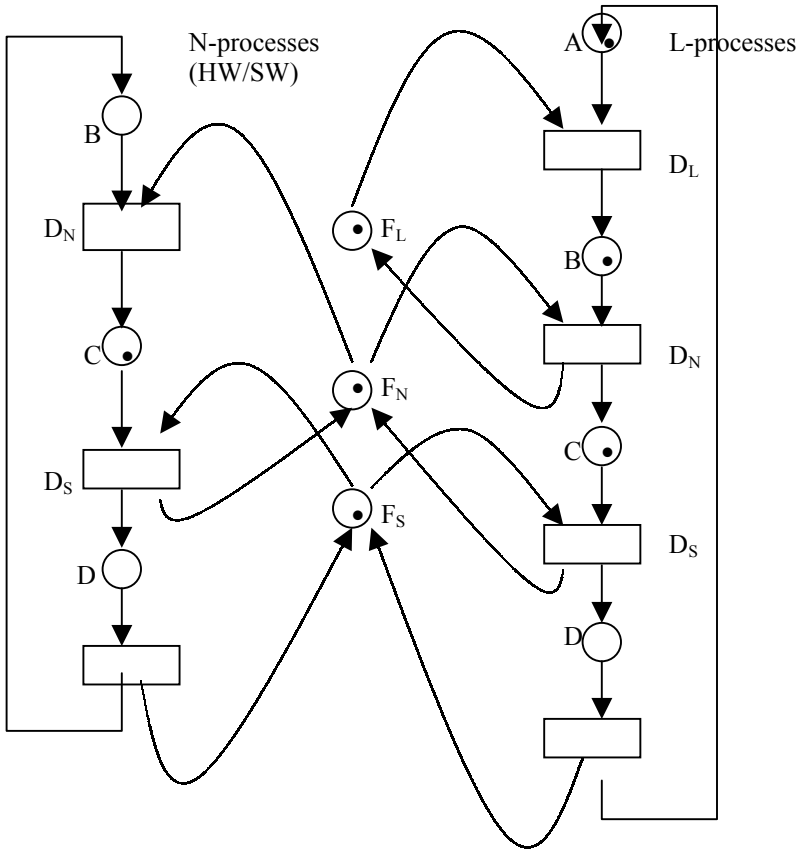



Figure 2.12 PT-net Describing the Processes in An Integrated Network

In Figure 2.12 we have to represent the two kinds of processes by two separate subnets – even though the N-process and L-process encounter failures in a similar way. This kind of problem is annoying for small problem, and it may be catastrophic for the description of a large network. Practical systems often contain components which are similar but not identical. Using PT-nets, these components must be represented by disjoint subnets with a nearly identical structure. So the practical use of PT-nets to describe real-world systems has demonstrated a need for more powerful net types to describe complex systems in a

manageable way. The development of high level Petri nets constitutes a very significant improvement in this respect. CP-nets (CPN) belong to the class of high-level nets.
The more compact representation has been achieved by equipping each token with an attached data value – token color. For a given place all tokens must have token colors that belong to a specified type. This type is called the color set of the place. The use of color sets in CPN is analogous to the use of types in programming languages.

A CPN consists of three different parts: the net structure (i.e. the places, transitions and arcs), the declarations and the net inscriptions (i.e., the various text strings which are attached to the elements of the net structure). CPN ML language is used for declarations in our study.

Now the system described in Figure 2.13 can be represented in a compact way by CPN as in Figure 2.14. A distribution of tokens on the places is called a marking. The initial marking is determined by evaluating the initialization expressions, i.e., the underlined expressions next to the places. In the initial marking (Figure 2.6) there is one (L, 0) tokens on A, B and C, while D has no tokens. Moreover, each of $F_L$, $F_N$, $F_S$ has one token. The marking of each place is a multi-set over the color set attached to the place. Multi-sets allow two or more tokens to have identical token colors. We shall also allow initialization expressions which evaluate to a single color c, and interpret this as if the value was 1’c (i.e., the multi-set contains one appearance of c).

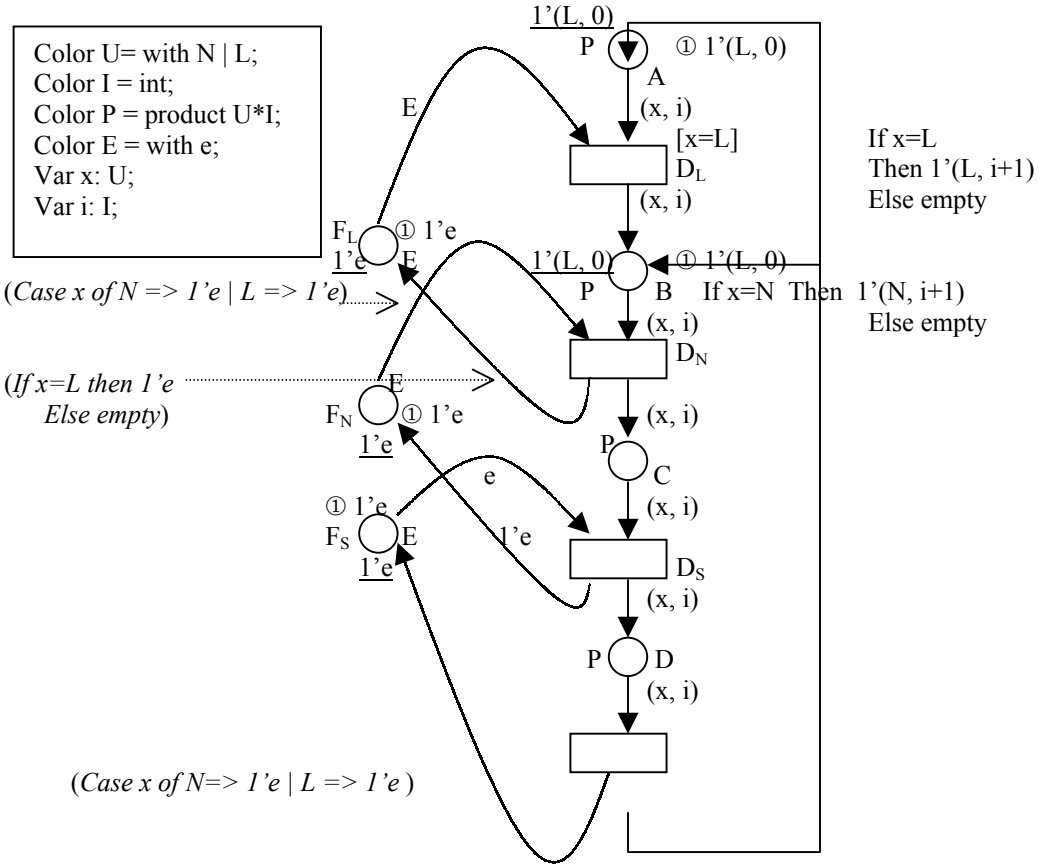



Figure 2.13 CPN Describing the Failure Modes in the Integrated Network

There are some arc expressions around transitions in Figure 2.13. These expressions have two variables, x and i, and from the declarations it can be seen that x has type U while i has type I, e is an element of the color set E while N and L are elements of U. x and i need to be bound to colors of the corresponding types (i.e., elements of the color sets U and I). One possibility is to bind x to N and i to zero: then we get the binding $b_1$ = <x = N, i = 0>. For each binding we can check whether the transition with that binding is enabled in the current marking. For the binding $b_1$ the two input arc expressions evaluate

to (N, 0) and 1'e, respectively. Thus we conclude that $b_1$ is enabled. CPN contains both case expressions and if-expressions to illustrate different possibilities, such as "*case x of N => 1'e | L => 1'e*". Expressions in Figure 2.6 with an italic style are just to show the choice functions, no special meaning in the specific system. More CPN ML knowledge can be referred to [94, 97].

From the above experiment, it is observed that the benefits achieved by using CPN instead of PT-nets, are very similar to those achieved by using high-level programming languages instead of assembly languages.

- Description and analysis become more compact and manageable because the complexity is divided between the net structure, the declarations and the net inscriptions.
- It becomes possible to describe simple data manipulations in a much more direct way by using arc expressions instead of a complex set of places, transitions and arcs.
- It becomes easier to see similarities and differences between similar system parts because they are represented by the same subnet.
- The description is more redundant and this means that there will be less errors.

Some kinds of errors become impossible or at least unlikely, e.g., it is difficult to add an extra state for the N-processes without considering whether the same should be done for the L-processes. It is possible to create hierarchical descriptions, i.e., structure a large description as a set of smaller CPN with a well-defined relationship.

## 2.6 Possibilistic Reliability Functions and Fuzzy Sets Theory

Classically, reliability theory has been based upon binary structure functions and probability theory. A binary structure function represents the deterministic relation between the component states and the system states, while probability theory is applied to develop the notion reliability of both components and systems.

Some obvious problems arise while applying this theory. A binary structure function allows only two states: a perfect functioning or a complete failure. The binary structure functions are too restrictive to model real life situations, since the concepts of failure or functioning are not always well defined or since a binary approach is too restrictive [81]. Hence, intermediate states must be allowed to describe the more complex systems. This is the topic of multistate structure functions that is closely related to fuzzy set theory since many real life problems simply cannot be represented by a dichotomous model.

By allowing intermediate states, we must extend the classical notion of reliability based on the probability of failure or functioning of a component or system. Some research showed that probability theory is not the only possible way of representing imprecision and uncertainty. Possibility theory and fuzzy set theory, e.g., provide useful alternatives to the probabilistic approach of reliability.

In classical reliability, probability theory is considered as the unifying model to represent uncertainty since classical reliability theory was developed at the early 30s and mainly after the WWII as an application of probability theory and quality control. Later on, the

reliability theory became a new, mainly a probabilistic field of interest. At that time, non-probabilistic uncertainty models were not available or at least not very popular. The confidence that the system will function properly at a certain level is classically defined in a probabilistic way, and leads to the well-known definition that the reliability of a system is the probability that the system functions during a certain time period.

On the other hand, some important deficiencies of the probabilistic approach became apparent in the early 60s. NASA developed alternative models to analyze the reliability aspects of the Saturnus V missile, since a classical approach failed. There were some reasons why a probabilistic approach was not successful. There was, e.g., an accumulation of errors due to the lack of sufficient statistical information about the failure aspects of the components, hence, there was an overestimation of the probability of failure. A qualitative approach was more appropriate. Since the introduction of fuzzy sets and possibility theory, new tools became available to model uncertainty. They are more qualitative by nature and can therefore be applied to situations where a quantitative approach is very unlikely or even impossible.

Several recent models to solve the problems mentioned about have been proposed based on fuzzy set theory. The fuzzy probabilities, the fuzzification of classical reliability function, and the combination of fuzzy states and fuzzy probabilities were introduced [82-84].

# CHAPTER 3

# PROBLEM FORMULATION

Network failures can arise in a couple of different ways. Failures may occur because the routing algorithm is unable to detect a functional route, although one exists. Failures may also happen if the flow control algorithm causes the network to be flooded with traffic, resulting in network failure due to overload. Both events are caused by software control of the network as protocols we usually mention, rather than by topological considerations.

Failures at a topological level can result from actions by intentional attack, natural disaster, or component wear-out. Intentional attack are purposefully selected to damage and inflict the network operation, comparing natural disasters are not. Typically damages on some portion of topology is in a small region but not in random. On the other hand, component wear-out is a random process and failures of each component are independent.

The network reliability and availability problem to be studied is focused on practical networks integrated with component systems where the software and hardware subsystems in nodes and hardware of transmission links are subject to independent

failures, additionally the 1:1 system redundancy initiatives deployed to improve the network high availability are also considered.

The problem needs to be formulated before proposing the approach. A stochastic network is a graph $G = (V, E)$, where $V$ and $E$ are the sets of vertices (node, $V$) and edges (link, $E$) of $G$. Each node, link, group, and the network is either operational or failed. Edge failures are mutually independent of each other with assumed or known probabilities. Nodes are mutually independent of each other with derivable probabilities. A node is operational if and only if both its contained software and hardware operate as intended. When a node fails, all links incident to the node also fail.

Usually nodes are subject to hardware and software failures while links are only related to the hardware problems. In practice, software such as control and communication protocols are stored in servers of the network. In some cases, hardware failures are induced by software failures. In such a situation, we assume that the hardware and software are in series inside a node, and fail independently. So the failure of a node results from the failure of the hardware part or the software part, or both. Software debug is assumed to be perfect, that is, debugging does not introduce new faults.

Notations are defined as following:

| | |
|---|---|
| $s, t$ | source, terminal nodes of node pair |
| $n, m$ | number of nodes, links in the network |
| $V_i$ , $E_j$ | node $i$, link $j$ in the network, where $i = 1, 2, \ldots n$, $j = 1, 2, \ldots m$ |

| | |
|---|---|
| $\alpha_i, \beta_j$ | operational probability of node $i$, link $j$ |
| $\alpha_{ih}$ | operational probability of hardware part in node $i$ |
| $\alpha_{is}$ | probability of software part in node $i$ functions as designed |
| $\varepsilon_i$ | utilization of software inside node $i$ |
| $h(t_i)$ | hazard function during the time $t_i$, between the ($i$-1)st and $i$th failure |
| $S_i, F_i$ | event $i$ which is successful, failed |
| \|S\|, \|F\| | number of successful events, failed events |
| $N_i, K_i$ | number of failed, operational links directed into node $i$ |
| $S_{i,j}, F_{i,j}$ | links with terminal node $j$ are operational, failed as specified by event $i$ |
| $R$ | node-pair reliability from $s$ to $t$ |

# CHAPTER 4

# APPROACHES FOR CALCULATING NETWORK RELIABILITY

## 4.1 Probabilistic and Deterministic Networks

A network $G = (V, E)$ consists of a set $V$ of nodes together with a set $E$ of edges, representing pairs of nodes. At any instant the elements of the network (nodes and/or edges) will be in either of two possible states, working or failed. In a deterministic network, it is considered that an adversary can successfully attack working elements, resulting their failure or inactivation. The failure of an edge means that it is removed from the network; while the failure of a node means that the node and all its incident edges are removed from the network.

In deterministic network models, the focus is typically on evaluating the worst-case performance of the network, in which the adversary intelligently chooses certain elements to render inactive, that would result in the maximum damage to the network. This type of network thus provides a conservative assessment of performance, and it would be partially appropriate in the design of robust systems.

On the other hand, it is assumed in probabilistic networks that, at any instant, elements fail randomly and independently of one another, according to certain known probabilities.

Specifically, each node $i$ has an associated reliability $p_i$ indicating the probability that it is operational, and each edge $k$ has a reliability $p_k$ which is the probability that it is operational. Thus at any instant the elements of the network fail independently with probabilities $q_i = 1 - p_i$ and $q_k = 1 - p_k$, respectively.

In these circumstances, one would be interested in assessing the average performance of the network, under the assumption of random (as opposed to malevolent) failures. It is also possible to allow for dependent failure modes, at the expense of added data-gathering requirements and increased subsequent computation. For example, the edges incident with a given node might be subject to certain common influences (such as weather, interference, or jamming), and these edges might therefore tend to fail together, rather than independently; or the failure of one edge might place additional stress on the other operating incident edges, making them more likely to fail.

Graph theory plays a key role in the analysis and design of reliable or invulnerable networks. According to Boesch [23], one can use a deterministic model that is called network vulnerability, contrasting to the usual probabilistic model for network reliability. Many different vulnerability criteria and the related synthesis results were reviewed. These synthesis problems are all graph external questions. Certain reliability synthesis problems can be converted to a vulnerability question. He distinguished between two types of models, summarized the relevant graph theoretic notions and then summarized the major results corresponding to each model.

## 4.2 Network Operations

Network reliability is concerned with the ability of a network to carry out a desired network operation. Therefore, an important first step is to identify necessary network operations.

The most common network operation is maintaining some connections or links between a source node $s$ to a target node $t$. *Two-terminal reliability* is defined as the probability that there exists at least an s-t path in a probabilistic graph $G$. In the directed case, the problem is usually called *s-t* connectedness.

The second most common operation in networks is broadcasting. We define the *all-terminal reliability* to be the probability that for every pair of nodes there is at least a path between. This is equivalent to the probability that there is at least one spanning tree in the graph. In a directed case, the reachability is the probability that there are paths from the source node to every other node.

The third and final one involves pair-wise communication of $k$ specified nodes, $2 \leq k \leq n$. the *k-terminal reliability* is the probability that for $k$ specified target nodes, the graph contains paths between each pair of the $k$ nodes. The directed analogue is called *s-t* connectedness.

## 4.3 General Approaches for Calculating the Reliability of Probabilistic Networks

There are several types of general approaches for calculating the reliability of probabilistic networks. Suppose that G = (*N*, *E*) is a directed network, having a distinguishable source node *s* and distinguishable destination node *t*. The nodes of G are assumed to be perfect, whereas the edges $k \in E$ are assumed to fail in a statistically independent fashion with known probabilities $q_k = 1 - p_k$. We will illustrate the general approaches with the *two-terminal* reliability $R_{st}(G)$ which is the probability of that there is a path of operative edges from *s* to *t* in *G*.

### 4.3.1 State-space Enumeration

The most fundamental method of calculating $R_{st}(G)$ uses state-space enumeration and dates back to Moore and Shannon [19]. It is a simple strategy that enumerates all states (all possible subgraphs), determines which are pathsets, and sums the occurrence probabilities of each pathset. Determining whether a state is a pathset is accomplished in general by using the supplied pathset recognition algorithm which employs standard path-finding or spanning tree methods.

Since each of the $m = |E|$ edges of G assumes one of two states, working or failed, the state of the network can be represented using 0-1 vector $\delta = (\delta_1, \delta_2, \ldots \delta_m)$. The *k*th component of $\delta$ equals 1 if edge *k* is working and is 0 if failed. Assuming edges fail independently, the probability of a given state $\delta$ is

$$p(\delta) = \prod_{k=1}^{m} p_k^{\delta_k} (1 - p_k)^{1-\delta_k}$$

Define the 0-1 variable $I_{st}(\delta)$, which equals 1 precisely when the sub-network of operational edges k (having $\delta_k = 1$) contains an *s*-*t* path. Then the two-terminal reliability is given by

$$R_{st}(G) = \sum_{\delta = D} I_{st}(\delta) P(\delta) \qquad (4.1)$$

where *D* is the set of all network states. Even though it's conceptually simple, the state-space approach is impractical because $|D| = 2^m$ and the computation time and cost increase exponentially with the network size.

We now illustrate the approach in a network with four nodes and five edges shown in Figure 4.1.

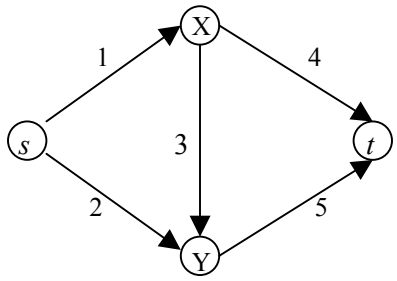



Figure 4.1 A Sample Bridge Network

It is obvious that the network contains a *s*-*t* path if at most one edge fails, or any two edges other than {1, 2}, {1, 5}, {4, 5} fail. On the other hand, for three or more edge failures, the network fails unless the failed edges are {1, 3, 4} or {2, 3, 5}. Thus the two-terminal reliability can be given as

$$R_{st}(\mathrm{G}) = p_1p_2p_3p_4p_5 + q_1p_2p_3p_4p_5 + p_1q_2p_3p_4p_5 + p_1p_2q_3p_4p_5 + p_1p_2p_3q_4q_5 +$$
$$p_1p_2p_3p_4q_5 + q_1p_2q_3p_4p_5 + q_1p_2p_3q_4p_5 + p_1q_2q_3p_4p_5 + p_1q_2p_3q_4p_5 +$$
$$p_1q_2p_3p_4q_5 + p_1p_2q_3q_4p_5 + p_1p_2q_3p_4q_5 + q_1p_2q_3q_4p_5 + p_1q_2q_3p_4q_5$$

Substituting $q_k = 1 - p_k$ into the above equation, and simplifying, we get,

$$R_{st}(\mathrm{G}) = p_1p_2p_3p_4p_5 - p_1p_2p_3p_5 - p_1p_2p_4p_5 - p_1p_3p_4p_5 + p_1p_3p_5 + p_1p_4 + p_2p_5$$

Although as many as 55 terms could have resulted from performing these substitutions, a good deal of cancellation occurred in producing the above expression.

Since only states $\delta$ with $I_{st}(\delta) = 1$ contribute to Equation (4.1), it is unnecessary to examine all states of $D$, except for those containing the above expressions. It is therefore appropriate to focus directly on the simple s-t paths $\{P_1, P_2, \ldots, P_k\}$ of $G$.

Define $E_i$ as the event that all edges in path $P_i$ operate. Then the two-terminal reliability is the probability that at least one such event occurs, or

$$R_{st}(\mathrm{G}) = \mathrm{P}(E_1 \cup E_2 \cup \ldots \cup E_k) \qquad (4.2)$$

The two-terminal network reliability can be alternatively formulated using the minimal *s-t* edge disconnecting sets, or cutsets of G. An *s-t* edge disconnecting set is minimal if it does not contain any other edge disconnecting set separating *s* and *t*. Indeed, suppose that the *s-t* cutsets are $\{C_1, C_2, \ldots, C_r\}$ and let $F_j$ be the event that all edges in cutset $C_j$ fail. Then the two-terminal unreliability $U_{st}(\mathrm{G})$ is given by

$$U_{st}(\mathrm{G}) = 1 - R_{st}(\mathrm{G}) = \mathrm{P}(F_1 \cup F_2 \cup \ldots \cup F_r) \qquad (4.3)$$

The events $E_i$ in Equation (4.2) are not in general disjoint, nor are the events $F_i$ in Equation (4.3). However, there are other standard methods for evaluating the probability of the union of the events.

Another way of viewing state-space enumeration emerges from the binary nature of the states assumed by each edge. Rather than fully specifying the states of all m edges at once, we can instead select a particular edge $e \in E$ and condition on the status of $e$, either perfect ($p_e = 1$) or failed ($p_e = 0$). We obtain a new system denoted $G/e$ in which edge $e$ is perfect in the first case, and another new system $G - e$ in which $e$ is failed for the second case. This produces the pivotal decomposition formula:

$$R_{st}(\mathrm{G}) = p_e R_{st}(G/e) + (1 - p_e) R_{st}(\mathrm{G} - e) \qquad (4.4)$$

This formula shows how reliability calculations for a given network can be decomposed into those for two smaller networks, $G/e$ and $G - e$. While conditioning, or factoring, in turn every possible edge just reproduces state-space enumeration, there are circumstances in which not all edges need to be considered for factoring. In fact, by judiciously selecting the edges for factoring, substantial computational saving can be achieved.

### 4.3.2 Inclusion-Exclusion

Using the principle of inclusion and exclusion, equation (4.2) can be expanded as

$$R_{st}(G) = \sum_{i} P(E_i) - \sum_{i<j} P(E_i E_j) + \sum_{i<j<l} P(E_i E_j E_l) - \ldots + (-1)^{k+1} P(E_1 E_2 \ldots E_k)$$

The intersection of event A and B is indicated by the juxtaposition of AB. Each term in this expansion is easy to calculate base on the independence assumption. However, there are $2^k - 1$ terms to appear, hence the computation time increases exponentially with the number of given paths.

For the sample network in Figure 4.1, there are three simple *s*-*t* paths.

$P_1$: 1-4 $P_2$: 2-5 $P_3$: 1-3-5

Thus, $P(E_1) = p_1p_4$, $P(E_2) = p_2p_5$, $P(E_3) = p_1p_3p_5$, $P(E_1E_2) = p_1p_2p_4p_5$, $P(E_1E_3) = p_1p_3p_4p_5$, $P(E_2E_3) = p_1p_2p_3p_5$, $P(E_1E_2E_3) = p_1p_2p_3p_4p_5$.

Application of the inclusion-exclusion method then produces the expression as follows,

$$R_{st}(G) = P(E_1) + P(E_2) + P(E_3) - P(E_1E_2) - P(E_1E_3) - P(E_2E_3) + P(E_1E_2E_3)$$
$$= p_1p_4 + p_2p_5 + p_1p_3p_5 - p_1p_2p_4p_5 - p_1p_3p_4p_5 - p_1p_2p_3p_5 + p_1p_2p_3p_4p_5$$

The topological formula of Satyanarayana and Prabhakar [34] is the most efficient method based on the inclusion-exclusion approach, although the number of terms in the reduced expression can still grow rapidly with the problem size. A reduced inclusion-exclusion formula for $R_K(G)$ holds in directed networks. Boesch et al. [35] discussed various combinatorial interpretations of the formula for $R_K(G)$.

### 4.3.3 Disjoint Product

Another way to calculate the probability of the union of events in Equation (4.2) is to decompose $E_1 \cup E_2 \cup ... \cup E_k$ into a union of events that are disjoint. Specifically we can express

$$R_{st}(\mathrm{G}) = \mathrm{P}(E_1 \cup E_2 \cup ... \cup E_k)$$

$$= \mathrm{P}\left(E_1 \cup \overline{E}_1 E_2 \cup \overline{E}_1 \overline{E}_2 E_3 \cup ... \cup \overline{E}_1 \overline{E}_2 \overline{E}_3 ... \overline{E}_{k-1} E_k\right)$$

where $\overline{E}_i$ denotes the complement of event $E_i$. Since the compound events above are pairwise disjoint,

$$R_{st}(G) = P(E_1) + P(\overline{E}_1 E_2) + P(\overline{E}_1 \overline{E}_2 E_3) + ... + P(\overline{E}_1 \overline{E}_2 \overline{E}_3 ... \overline{E}_{k-1} E_k)$$

This disjoint-products method involves adding only $k$ probabilities. However, the calculation of each constituent probability is generally involved. It is also important to emphasize that the efficacy of this method can be highly dependent on the specific ordering given to the events $E_i$.

A number of methods [36-37] have been proposed to carry out the disjoint-products method, varying in their specific details but following the overall strategy. Typically the paths $P_i$ are first ordered by non-decreasing length and then processed in turn to generate a number of terms disjoint with one another and those previously generated. In general, the number of generated terms can grow rapidly with the number of given paths $k$. In particular, the disjoint-products method can be carried out efficiently, in terms of $k$, for the all-terminal reliability problem in directed networks (a nondenenerate linear system). No such efficient method is known for calculating the two-terminal reliability problem.

### 4.3.4 Factoring

The inclusion-exclusion and disjoint-products techniques are based on a given enumeration of the *s*-*t* paths. The factoring method does not require knowledge of these paths but instead concentrates on the state of an individual edge. Application of the pivotal decomposition Equation (4.4) creates two sub-problems with smaller size. If the decomposition were simply reapplied to each such sub-problem, the approach would not be better than state-enumeration. Crucial to this approach is the possibility that certain of generated sub-problems might be reduced in size using simple probabilistic rules.

Some basic rules of reduction are presented now. Two edges $e = (i, k)$ and $f = (i, k)$ joining the same two nodes in a directed network $G$ are called parallel edges. A parallel reduction replaces two parallel edges, having probabilities $p_e$ and $p_f$, by a single edge having probability $1 - (1 - p_e)(1 - p_f) = p_e + p_f - p_e p_f$. Two edges $e = (i, j)$ and $f = (j, k)$ are called series edges if these are the only two edges incident with node $j$. If $j \neq s, t$ then a series reduction replaces the two series edges by a single edge having reliability $p_e p_f$. Figure 4.2 illustrates these two reliability-preserving reductions, which are valid in view of the independence of edge failures. Also illustrated is a more general two-neighbor reduction, applicable if $j \neq s, t$.

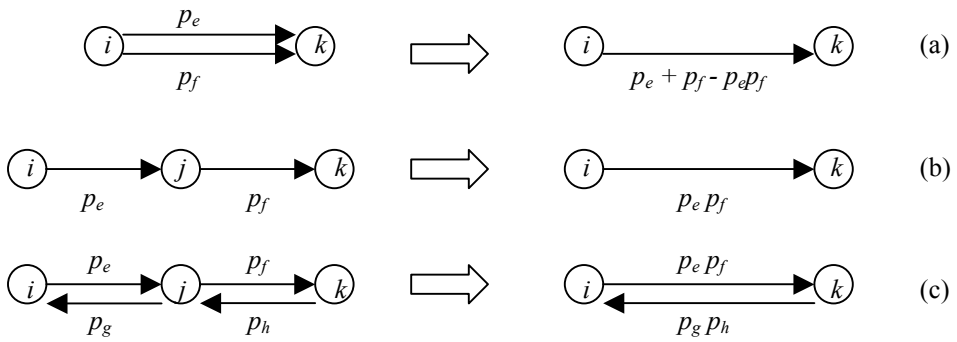


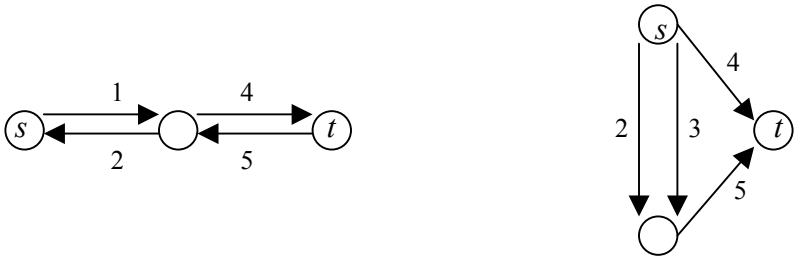

Figure 4.2. Probabilistic Rules of Reduction

A network $G$ is two-terminal series parallel if it can be reduced to a single edge $(s, t)$ by repeatedly applying series and parallel reductions. In such a case, the two-terminal reliability is simply the reliability appearing on the final edge, and efficient algorithms exist for identifying and carrying out the appropriate reductions. More generally, the application of series and parallel reductions to $G$ will leave a network more complex than a single edge. At this point, an edge can be selected for conditioning and the pivotal decomposition formula can be applied, yielding two new sub-problems. Series and parallel reductions are applied to these sub-problems for as long as possible, at which point pivotal decomposition can again be invoked. This alternating strategy of pivotal and applying reliability-preserving reductions constitutes the factoring algorithm.

For a directed network $G$, factoring on an edge $e$ out of $s$, or into $t$, is especially helpful. The system $G/e$ will have a topological interpretation, since it is the network obtained from $G$ by deleting edge $e$ and merging its endpoints. While Equation (4.4) remains valid for any edge, unless the choice of edge for factoring is suitably restricted, $G/e$ will not necessarily be equivalent to the network obtained from $G$ by contracting the edge. This is

clearly seen in the network of Figure 4.1, since contraction of edge 3 would produce the spurious path 2-4 in Figure 4.3(a). On the other hand, contraction of edge 1 produces the series-parallel network shown in Figure 2.3(b) and its reliability is easily calculated as

$$R_{st}(G) = (p_2p_5 + p_3p_5 - p_2p_3p_5) + p_4 - (p_2p_5 + p_3p_5 - p_2p_3p_5)\, p_4$$




(a) (b)

Figure 4.3. Contraction of an Edge in Fig 4.1, Using (a) $e$ = 3 and (b) $e$ = 1

Also $G - e$ is accurately represented by the network of Figure 4.1 with edge 1 removed. Since edge 3 and 4 are then irrelevant, they can be removed and $R_{st}(G - e) = p_2p_5$. As a result of factoring on a single edge the two-terminal reliability of $G$ is determined as

$$R_{st}(G) = p_2\, R_{st}(G/e) + (1 - p_1)R_{st}(G - e)$$

$$= p_1p_4 + p_2\, p_5 + p_1p_3p_5 - p_1p_2p_3p_5 - p_1p_2p_4p_5 - p_1p_3p_4p_5 + p_1p_2p_3p_4p_5$$

The factoring approach was first applied to directed networks by Nazakawa [38]. Reliability algorithms for directed networks that incorporate factoring, together with probabilistic reduction rules, were implemented [39-40]. Johnson [41] and Wood [42] discussed the application of the factoring approach to a variety of network reliability

problems, in particular the *k*-terminal and all-terminal reliability problems for undirected networks.

### 4.3.5 Fault Tree Analysis

The technique of Fault Tree Analysis (FTA) for the estimation of the frequency of occurrence of an event was formalized in 1962 at Bell Laboratories.

FTA is a very useful and popular method for analyzing complex system reliability. The fault tree itself is a graphic representation of the Boolean failure logic associated with the development of a particular system failure (the TOP event) to basic failures (primary events). For example, the TOP event could be the failure of a nuclear power plant guidance control system during its operation with the primary events being the failures of individual guidance control system components.

FTA can be a valuable design tool. It can identify potential accidents in a system design and can help eliminate costly design changes and retrofits. FTA can also be a diagnostic tool. One can predict with it the most likely causes of system failures in the case of system breakdown.

The fault trees are a special case of decision trees and contain logical gates, (for example, AND, OR, NOT, NOR, NAND, *k-out-of-n*) and symbols of top end primary events. The goal of fault tree construction is to model the system conditions that can result in the undesired event. Before construction of the fault tree, a thorough understanding of the system is acquired. In fact, a system description should be a part of the analysis

documentation. The analyst must carefully define the undesired event under consideration, called the "top event".

FTA can involve the following steps:

- System definition
- Fault tree construction
- Qualitative analysis
- Quantitative analysis

System definition combines the analysis objectives with information about the systems. The analysis objectives guide the selection of TOP events. Boundary conditions define physical and analytical bounds associated with a TOP event and, together with a statement of the TOP event, constitute a problem definition.

Fault trees are constructed for each of the TOP events based on the system definition step. Operator failures are included in the fault trees. The potential for operator acts of commission is not explicitly included in the fault trees but is indicated in the appropriate basic component failures.

The qualitative analysis includes determining system failure modes-called minimal cut sets-for each fault tree. The minimal cut sets are used as input to the quantitative analysis, and they provide structural importance information about basic events (component and human failures). The most structurally important basic events are those that are one-event

cut sets; the next most important basic events are those in the largest number of two-event cut sets, and so forth.

In many instances, it is not necessary to determine all minimal cut sets for a TOP event. If there are many low-order minimal cut sets (cut sets containing small numbers of basic events), these cut sets will usually dominate the system failure probability, and higher-order cut sets do not need to be determined.

The quantitative analysis step includes determining TOP event reliability characteristics from the minimal cut sets and the component failure characteristics assuming that all-component failures and repairs are independent. Four quantitative reliability characteristics were of interest in the utility system study:

- System unavailability
- Expected number of system failures
- Average system downtime
- Component importance

The system unavailability at a given time is the probability that the system is in the failed state at that time. The expected number of system failures is the expected number of times that a system failure will occur over a time interval. The average system downtime (for repairable systems) is the quotient of system unavailability and system failure rate. component importance estimates the fraction of time that a component failure is contributing to system failure, given the system is failed.

## 4.4 Computational Complexity of Reliability Analysis

Reliability analysis problems are more closely aligned with counting problems where the objective is to determine the number of configurations of a particular type. The minimum cardinality pathset problem associated with the *k*-terminal problem is the problem of finding a minimum cardinality Steiner Tree. Rosenthal [24] firstly showed that reliability analysis for *k*-terminal networks are all *NP-hard*. The minimum cardinality pathset problem associated with the 2-terminal problem is the problem of finding a shortest (*s, t*) path. It was first proved by Valiant [25] that the functional, rational, and point estimate reliability analysis problems are all *NP-hard* for the 2-terminal networks. For all-terminal measure it is necessary to analyze direct and undirected networks separately. The minimum cardinality cutset problem is the problem of finding a minimum cardinality *s*-directed cut. Provan and Ball [26] proved that the reliability analysis problems for the directed and undirected all-terminal measure are *NP-hard*.

A standard source for information on the computational complexity of algorithms is the book of Garey and Johnson [74]. More specific information on the complexity of network reliability problems and NP-complete problems can be found from [4, 24-25].

The usual definition of NP employs a model of nondeterministic computation, the nondeterministic Turing machine. Turing machines that halt either accept or reject their input; however, there may be a number of different nondeterministic choices that would lead to acceptance. For this reason, Valiant [76, 77] explored the extension to counting Turing machines, which act just like nondeterministic Turing machines, but upon

acceptance print the number of different computations which would lead to acceptance. Then #P (read "sharp P" or "number P") is the class of functions which can be computed by counting Turing machines in polynomial time. Naturally the counting version of any problem in NP is in #P; however, the counting Turing machine is apparently a nontrivial extension of the nondeterministic Turing machine, as there is no obvious way to produce the number of accepting computations just knowing the existence of one.

Complexity results can be obtained by transforming known NP-complete problem and #P-complete problems into the reliability problems.

# CHAPTER 5

# MODELING RELIABILITY OF INTEGRATED NETWORKS (MORIN)

## 5.1 MORIN Method

AGM has been rigorously proved as a corollary of the general theorem on complex system decomposition. Some other self-proclaimed more efficient algorithms are derived from it. The AGM method may be extended to solve problems in integrated systems where the software in a node has a constant failure rate [2]. However the computational time increases exponentially with the number of links. Another explicit method namely NPR/T [7], which was derived from AGM, is much simpler and more direct, and the computational time increases linearly with the number of links. But this method can yield incorrect results in some cases involving undirected networks [6]. At any rate, neither method covers network reliability problems when software failure follows different distributions.

The AGM method considers each link in the network (with failure-prone links and nodes) as a series combination of a perfect node and the link with modified reliability. However, the computing time increases exponentially with the number of links.

The approach for MOdeling Reliability for Integrated Networks (MORIN) adopts the strategy of replacing a network having unreliable nodes with an equivalent network having completely reliable nodes except the source node *s*. Considering link *i* and its terminal node *j*, the link in the equivalent network has a modified reliability $\alpha_j\beta_i$. In the equivalent network, the failures of all links are not necessarily *s*-independent, but failures of a link and other links that are connected to uncommon terminal nodes are still independent. For each node *j* (in event $S_i$) except the source node *s*, group its incoming directed links, and then compute *R* without Boolean simplification.

$$P\{S_i\} = \alpha_{sh}\alpha_{ss}\prod_{j=1}^{n-1} P\{S_{i,j}\} \tag{5.1}$$

where $S_{i,j}$ is operational links 1, 2, … $K_j$ directed into node *j* on event tree *i*, then

$$P\{S_{i,j}\} = \alpha_{jh}\ \alpha_{js}\prod_{i=1}^{K_j}\beta_i \tag{5.2}$$

If there are no links directed into node *j* specified by $S_i$, then $P\{S_{i,j}\}=1$. Let links 1, 2, …$N_j$ directed into node *j* be specified as failed and links $N_j$+1, $N_j$+2, … $N_j$+$K_j$ be specified as operational, then

$$P\{S_{i,j}\} = \alpha_{jh}\alpha_{js}\prod_{i=1}^{N_j}(1-\beta_i)\prod_{i=N_j+1}^{N_j+K_j}\beta_i \qquad \text{for } K_j \geq 1 \tag{5.3}$$

Let $K_j = 0$, then links 1, 2, … $N_j$ have failed in the equivalent network if and only if node *j* has failed and all $N_j$ links are operational, or all $N_j$ links have failed and node *j* is operational, or both node *j* and all $N_j$ links failed. Since the probability expression for

node $j$ does not reflect the fact that the failure of this node thereafter brings with its failures of links incident to this node, then:

$$P\{S_{i,j}) = 1 - \alpha_{jh}\alpha_{js} + \alpha_{jh}\alpha_{js}\prod_{i=1}^{N_j}(1-\beta_i) \qquad \text{for } K_j = 0 \qquad (5.4)$$

Since the $S_i$ are mutually exclusive events, the node-pair reliability is the summation of the probabilities of all success disjoint events, thus

$$R = \sum_{i=1}^{|S|} P\{S_i\} \qquad (5.5)$$

As showed above, the MORIN approach can be summarized as follows

- Find all mutual exclusive disjointed path set from the source node to sink node of the corresponding network, denoted as event trees $\{S_1, S_2, \ldots S_i\}$
- On each event tree $S_i$, for each node $j$ except the source node $s$, group its incoming directed links specified by $S_{i,j}$
- Denote $S_{i,j}$ as operational links 1, 2, … $K_j$ directed into node $j$, then

$$P\{S_i) = \alpha_{sh}\alpha_{ss}\prod_{j=1}^{n-1} P\{S_{i,j}\}$$

- Compute the $P\{S_{i,j}\}$ by considering failed and operational links for node $j$
- Combine above four steps and the Equation (5.1)(5.3)(5.4)(5.5) to get the reliability of entire network.

The pseudo-codes of MORIN can be presented as follows:

1. **MORIN_Events** (G, *s*, *t*)

// find all event trees $\{S_1, S_2, \ldots S_i\}$

// where source node is *s*, sink node is *t* and G = (V, E)

a. Initialize the network model

d(*s*) ← 0 : π(*s*) ← NIL // node *s* is the source node

S(*i*) ← {*s*} // Each event tree *i* includes source node *s*

Path_Set(*i*) ← NIL // Path-set is empty in event *i*

Q ← {*s*}

For each node *u* ∈ V[G] – *s*

Do d(*u*) ← ∞ // d(*u*) is the distance from *u* to *s*

π(*u*) ← NIL // π(*u*) is the predecessor node of *u*

color(*u*) ← white // node *u* has the not been discovered

b. Iterations

While Q ≠ NIL

Do *u* ← Head(Q)

For each *v* ∈ Adj(*u*)

Do if color(*v*) = white

then Path_Finding(*v*)

if π(*t*) = *v* // A *s*-*t* path is found

then S($i$) ← S($i$) + $v$

Path($i$) ← Path_Set($i$)

$i$ ← $i$ + 1

Path_Finding($v$)

color($v$) = gray

d($v$) ← d($u$) + 1

π($v$) ← $u$

Path_Set($i$) ← Path_Set($i$) + ($u$, $v$)

for each $w$ ∈ Adj($v$)

Do if color($w$) = white

then π($w$) ← $v$

Path_Set($i$) ← Path_Set($i$) + ($u$, $v$)

Path_Finding($w$)

Color($v$) = black

Q ← ENQUEUE(Q, $v$) //Add $v$ to head of the Queue

2. **Event_RCal** [S($i$)]

// Calculate the network reliability R based on generated event trees/path sets and

// reliability of each node and link along the event paths.

R = 0

for each path of path_set ($i$) on event tree S($i$)

$S_{i,j} \leftarrow$ group incoming directed links of node *j* on event *i*

$P(S_{i,j}) = 1$ // if $S_{i,j}$ does not specify any links directed into node *j*

While node Queue of $S_i \neq$ NIL

For all operational links into node *j*

$$P_o(S_{i,j}) = \alpha_j \prod_{i=1}^{K_j} \beta_i$$

For all failed links into node *j*

$$P_f(S_{i,j}) = (1-\alpha_j) + \alpha_j \prod_{i=1}^{n_j} (1-\beta_i)$$

$$P(S_i) = \alpha_s P_o(S_{i,j}) P_f(S_{i,j})$$

DEQUEUE (Q, *j*) // remove node *j* from the node queue of event S(*i*)

$R \leftarrow R + P(S_i)$

Prior to designing or evaluating the reliability/availability a network or an end-to-end solution, it is essential to model the reliability/availability of corresponding systems that normally comprise of hardware subsystems and software subsystems and are usually configured under a complex architecture. Additionally, redundancies at various levels (such as chipset level, board level, system box-level) are typically deployed in complex systems to achieve high availability (HA) in industry to meet practical application demands and requirements. This type of issues can be addressed by the simplified methodology and modeling tool (SAMOT) introduced in Chapter 6.

# CHAPTER 6

# SIMPLIFIED NETWORK AVAILABILITY MODELING

This chapter proposes a simplified methodology that incorporates Markov analysis and Reliability Block Diagram methodologies to model and analyze the availability of a typical end-to-end solution consisting of multiple complex component systems, where the failure of each component system is attributed to software failures and hardware failures. The methodology and computational tool - Simplified Availability Modeling Tool (SAMOT) is introduced. The application of SAMOT to 1:1 system redundancy, which is common in the networking industry, is the focus of this study. The end-to-end availability is modeled and computed based on the corresponding signaling path and bearer path since the paths can transverse through different component systems. It is observed that SAMOT is very accurate (compared with the Markov analysis) when applied to 1:1 redundant systems under various system parameter sets with high switchover coverage.

## 6.1 Introduction

High availability (HA) with its attendant higher requirements for system performance has increasingly become an important feature for suppliers of computer network equipment to communication service providers. Usually system failures are attributed to its hardware components or/and software components. The algorithms and approaches of modeling

and analyzing the availability of a communication network comprised of numerous, complex topology systems is the subject of much research [119]. However, very few HA modeling tools for complex networks are commonly accepted and applied in industry. A number of vendors have provided some commercial software applications (Relex[1], SelfReliant[2], MEADEP[3], SHARPE[4], RealSoft[5], etc.) for reliability modeling and analysis of complex systems. But adequate training and relevant experience in corresponding fields are required, in addition to the software license fee or purchase cost.

This chapter introduces a simplified interactive modeling tool that integrates Markov analysis and Reliability Block Diagram (RBD) methodologies for computing the availability of a typical end-to-end network solution where a 1:1 system-level redundancy is installed in some component systems. The Markov analysis is approximated by the Defect Per Million (DPM) model [116], and the RBD method is implemented by SHARC [117].

Definitions

*DPM* (*defects per million*): the number of calls lost per million calls attempted. It consists of two elements – call-blocked DPM and call-dropped DPM. To complete a communication transaction, the network must establish some paths (not necessarily physical circuits), e.g. a signaling path and a bearer path for voice packets, a signaling

[1] Relex is the registered trademark of Relex Software Corporation.
[2] SelfReliant is the registered trademark of GoAhead Software Inc.
[3] MEADEP is the registered trademark of SoHaR Inc.
[4] SHARP is the registered trademark of
[5] RealSoft is the registered trademark of RealSoft Pte Ltd.

path and a data path for data packets. Usually when a call is blocked, subscribers cannot make new calls due to the fact that there is at least one failure along the signaling path; whereas when an existing call is dropped, at least one failure occurs along the bearer path of the network.

$DPM = (1 - Availability) \times 10^6$

$Total\ DPM = DPM_{call\text{-}blocked} + DPM_{call\text{-}dropped}$

$$= \frac{(Number\ of\ calls\ blocked + Number\ of\ calls\ dropped) \times 10^6}{Number\ of\ calls\ attempted}$$

*End-to-end availability*: the probability that a customer can complete the communication to its destination. Since a signaling path and a voice path as well as a data path may pass through different network components, the end-to-end availability for each type of path can vary and therefore needs to be identified and studied at the path level.

*1:1 Redundancy*: there is one redundant unit for every unit that is required for full operation. Redundancy can improve availability by orders of magnitude while keeping the MTBF and MTTR of each unit the same. The effectiveness of redundancy is highly dependent on the switchover coverage and switchover time.

*Switchover Coverage*: the probability that a failure is success-fully detected, isolated, and recovered by a higher-level fault-management mechanism. In case of active/standby

redundancy, switchover coverage is dependent on the fault detection on the active side, the fault detection on the standby side, and the reliability of the switching mechanism.

Switchover coverage = Active fault coverage × Standby fault coverage

where *active fault coverage* is the probability of detecting a fault on the active side as well as having the switching mechanism operational at the same time; *standby fault coverage* is the probability of detecting a fault on the standby side. In the case of load-sharing redundancy, the switchover coverage is dependent upon only one fault-detection coverage because there is no inactive standby side.

*Switchover Time*: the time from when a failure is detected in an operating component to the time when the affected traffic is switched over to the redundant component.

More detailed definitions can be obtained in [116, 118].

## 6.2 Problem Description

A typical voice-over-internet protocol (VoIP) solution includes different functional segments -- access equipment, aggregation device, core router, LAN switch, edge system, etc. -- as shown in Figure 6.1. Each segment can encompass one or more systems. The end-to-end (signaling, voice, or data) traffic has to pass through most (if not all) segments to complete the transmission. The customer premium equipment (CPE) is usually located at customer side and its availability is affected by many non-system-reliability factors

(such as, process-related failures, human errors); thus, it is not considered in the end-to-end availability.

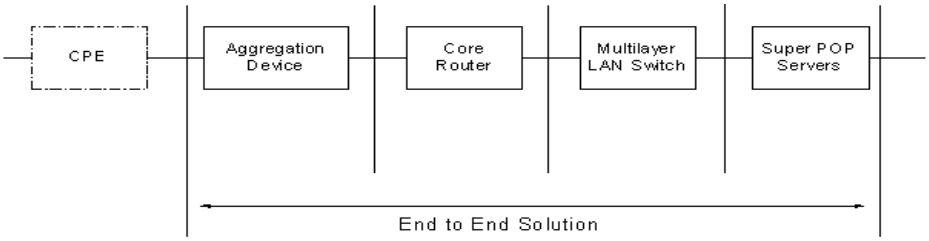



Figure 6.1. Segments of A Typical VoIP Solution

The end-to-end availability is determined by availabilities of component systems and network links along a given path. Furthermore, the system availability is attributed to the availability of system hardware and software, configuration, fault management mechanisms, and operation, administration and maintenance (OA&M). System hardware usually consists of an egress line card, an ingress line card, a chassis, processor card, dual power supply, and some other feature cards. System software normally includes the operation system software running on server platform or processor card and application software running on processor card or feature cards, depending on the specific system configuration. The fault management function can be performed by the monitoring/alarm system, online diagnosis system, etc. The planned outage comprises of software upgrades and hardware upgrades in this discussion.

Board-level and system-level redundancy can be deployed to improve the system and network availability. The system redundancy effectiveness [116] is mainly determined by

the redundancy type (active/standby or load-sharing), 1:1 or 1:N redundancy, switchover coverage, and switchover time.

Figure 6.2 illustrates the RBD of a sample system.

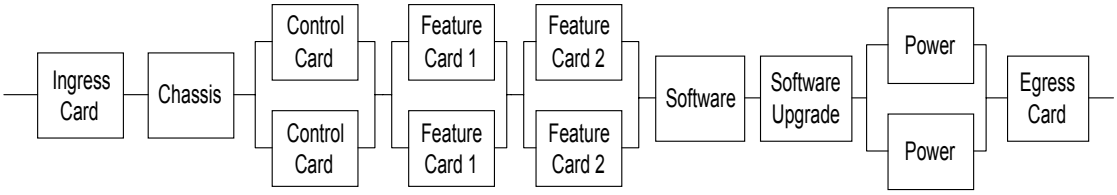



Figure 6.2. Reliability Block Diagram of A Sample System

The proposed modeling tool is to depict and predict the availability for the signaling path and bearer path of a typical network solution comprised of software-hardware systems with 1:1 redundancy at the box-level, considering both un-scheduled outages and scheduled outages.

## 6.3 Methodologies and Tools

### 6.3.1 Common Methodologies

The Markov modeling method is advantageous in terms of capturing the component failover behavior and fault coverage probability with states and state transitions. However the Markov modeling tool may be difficult to apply in the field. It can be complicated and computationally intractable when a system or network has a complex topology. RBD is one of the most commonly used methods in modeling serial-parallel system reliability. But it does not have the power to handle large networks with a complex topology.

### 6.3.2 Commonly-used Tools

The DPM model and SHARC are two practical tools for modeling system and network availability in industry. The DPM model was originally created to approximate the Markov method for calculating the availability of a network with a serial-parallel topology. Since software and hardware components of the redundant systems can have very different availability metrics such as MTBF, MTTR, switchover time and planned outages, the DPM modeling tool is not capable of taking the system box level redundancy schemes into consideration. The SHARC [117] applies the RBD method to compute the availability metric of a simplex system, however it is not capable of identifying the unavailability (downtime) contributed by the switchover time and imperfect switchover coverage for a redundant system. So an improved reliability block diagram (IRBD) is created, where several blocks are added to describe the switchover coverage and switchover time for active/standby redundant systems.

### 6.3.3 SAMOT Tool

The SAMOT calibrates and integrates the above two methodologies/tools (Markov/DPM and RBD/SHARC) and incorporates the availability design parameters into two inter-active modules [119] to model the end-to-end network availability. A sample network solution architecture (as shown in Figure 7.5), where each Super POP element deploys the 1:1 system redundancy, will be studied in Section 7.2.

The SAMOT interactive tool consists of a Main module and a Redundancy module. Each module is a separate spreadsheet file, which provides some input and act as output of the other file. The Main module models the availability of all component systems of the network, with each system on one sheet. If there is redundancy involved, the availability of the redundant systems is computed on the same sheet with input data categorized into planned outage and unplanned outage from the Redundancy module. The Main module calculates the availability of various end-to-end network paths as well.

The Redundancy module models the 1:1 redundant system availability by approximating the unplanned and planned outages resulted from major hardware and software failures. The output of the Redundancy module is the input of the Main module when calculating the availability of redundant systems. The Main module calculates the unplanned outage of hardware and software, and the planned outage of hardware and software of a single system as the input of the Redundancy module when corresponding system redundancy is involved. Figure 6.3 illustrates the interactive relationship between the two modules.

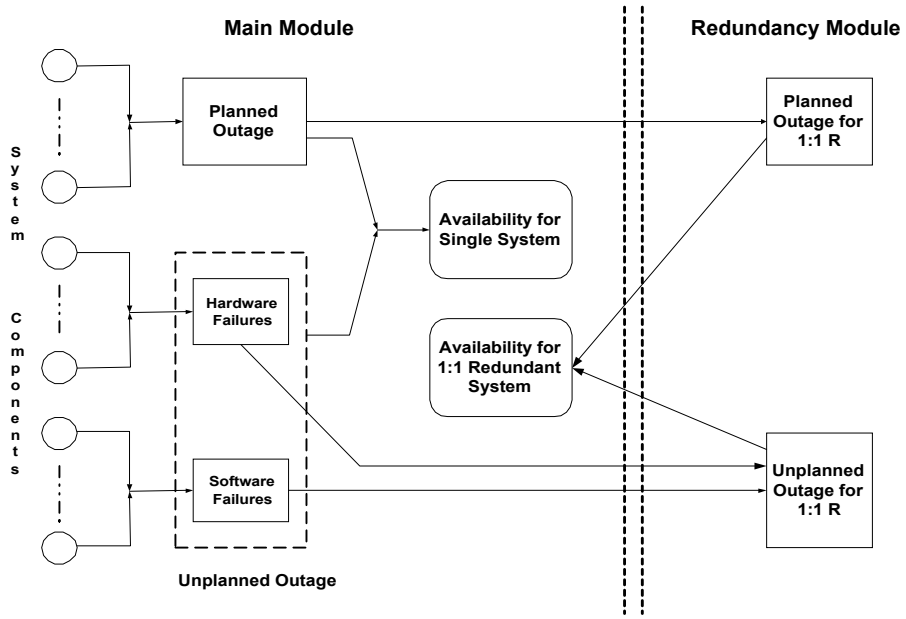



Figure 6.3. Interactive Modules in SAMOT

Since the hardware and software usually have quite different MTBF and MTTR availability attributes, their failures need to be considered separately. The IRBD in Figure 6.4 captures the major failure modes of the 1:1 redundant hardware-software systems. Those failure modes and parameters need to be preliminarily determined by design engineers or users of the tool before being applied in the SAMOT tool.

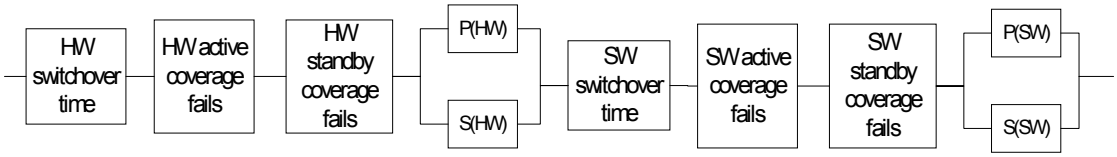



Figure 6.4. IRBD for 1:1 R in SAMOT's Redundancy Module

In Figure 6.4, the first four blocks illustrate the hardware failure modes for the 1:1 redundant systems.

- The “HW switchover time” block reflects the short duration outage that results from the switchover.
- The “HW active coverage fails” block depicts the system outage when the system fails to detect hardware failure on the active side or successfully detects the hardware failure on the active side but fails to switch over to the standby side.
- The “HW standby coverage fails” block describes the outage when an active side hardware failure is detected and traffic is being switched to the standby side, but the standby side hardware has failed and remained undetected.
- The parallel “P(HW)” and “S(HW)” blocks are to model the hardware system in the primary unit and secondary unit (sometimes called active and standby unit) with perfect coverage and Zero switchover time. The system outage happens when hardware on both sides fail.

Note: The standby coverage failure may not bring network outage immediately, should be in the protection path with S(HW) block. SAMOT adopts the IRBD in Figure 6.4 to simplify the approximated computation.

The software failure modes are taken into account similarly.

The manual failover tests can be considered to reduce outage from the standby coverage failure and improve the redundancy effectiveness. The impact of this change is trivial under the following experimental availability parameter settings.

Markov analysis is capable of exhaustively enumerating the failure states and their transitions; it is used to verify the correctness and accuracy of the Redundancy module of SAMOT for modeling the availability of a 1:1 R system.

Figure 6.5 is the Markov failure state transition diagram for the 1:1 redundant system. Among the 13 major states of the 1:1 redundant system, State 2, 4, 5, 6, 10, and 11 (double circled) represent failure modes. The symbol on each arc connecting one node to the other is the transition probability between the two states.

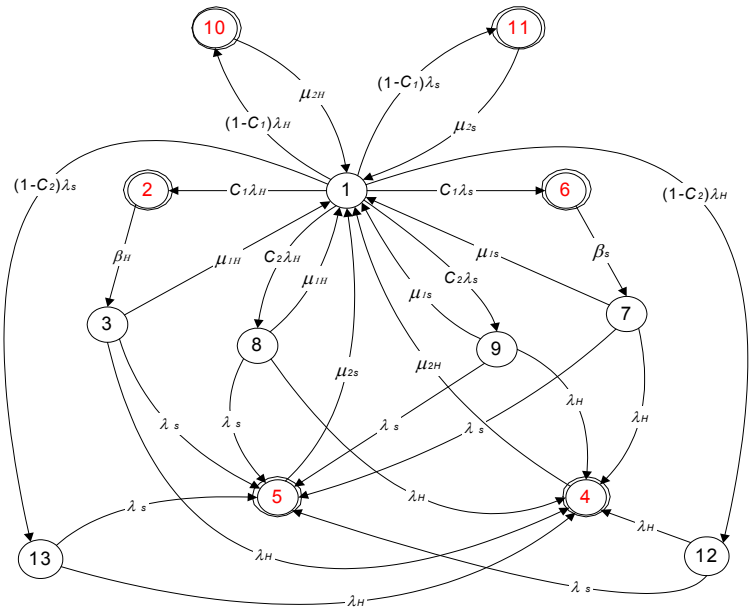



Figure 6.5. Markov Diagram for Failure Mode Transitions of 1:1 Software-hardware System Redundancy

Variables

$c_1$ = Coverage factor for active unit

$c_2$ = Coverage factor for standby unit

$\lambda_H$ = Hardware failure rate of individual unit

$\lambda_s$ = Software failure rate of individual unit

$\beta_H$ = Hardware switchover rate from active to standby

$\beta_s$ = Software switchover rate from active to standby

$\mu_{1H}$ = Hardware repair rate of non-service-affecting failures

$\mu_{1s}$ = Software repair rate of non-service-affecting failures

$\mu_{2H}$ = Hardware repair rate of service-affecting failures

$\mu_{2s}$ = Software repair rate of service-affecting failures

State Descriptions

| | |
|---|---|
| 1 | All hardware work |
| 2 | Hardware of the active unit failed, detected |
| 3 | Hardware of the standby unit has taken over |
| 4 | Hardware of 2nd unit failed while recovering the failed unit |
| 5 | Software of 2nd unit failed while recovering the failed unit |
| 6 | Software of the active unit failed, detected |
| 7 | Software of the standby unit has taken over |
| 8 | Hardware of the standby unit failed, detected |
| 9 | Software of the standby unit failed, detected |

| | |
|---|---|
| 10 | Hardware of the active unit failed, can not switch to standby |
| 11 | Software of the active unit failed, can not switch to standby |
| 12 | Hardware of the standby unit failed, undetected |
| 13 | Software of the standby unit failed, undetected |

# CHAPTER 7

# COMPUTATIONAL EXPERIMENTS

To demonstrate the applications of the proposed MORIN and SAMOT approaches and techniques for reliability and availability analysis of integrated networks, this chapter contains some computational experiments and results.

## 7.1 MORIN Examples

The two-terminal communication (e.g. communicating from a source node to a target node) is the most common network operation. The *k*-terminal reliability and all-terminal reliability problems can be derived from the two-terminal reliability problems. To demonstrate the MORIN approach, two-terminal reliability examples are used.

### 7.1.1 Sample Network 1

Figure 7.1 is an example of a typical directed bridge network. Nodes 1 and 4 are the source and terminal nodes respectively. The two black dots inside each node represent the corresponding hardware component and software component of the node.

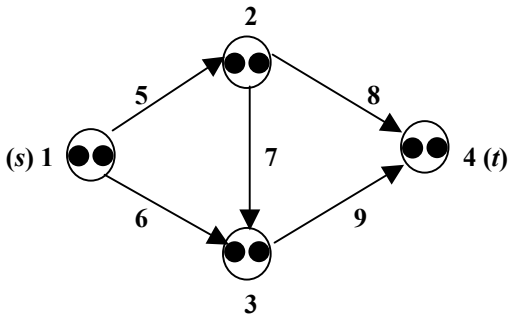



Figure 7.1 Sample Network 1

The *s*-*t* reliability can be obtained with the 4 success events, as shown in Figure 7.2:

$$S_1 = 58, \quad S_2 = \overline{5}69, \quad S_3 = 56\overline{8}9, \quad S_4 = 5\overline{6}7\overline{8}9,$$

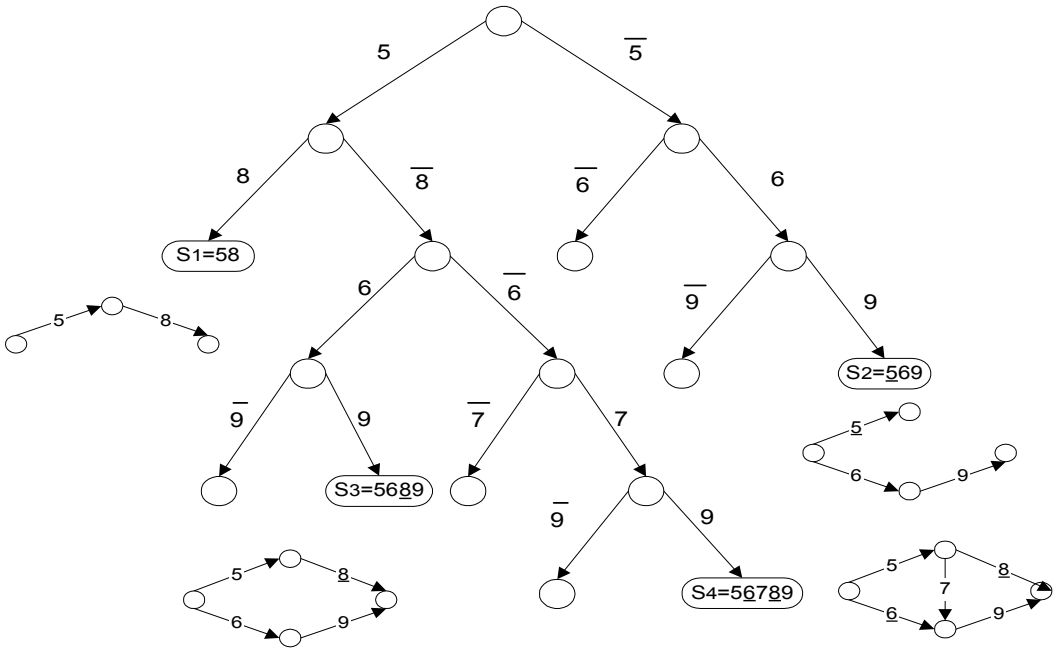



Figure 7.2. Event-Tree Generated by the MORIN Algorithm for Sample Network 1

Thus the symbolic expression of the reliability can be presented as,

$$R = \sum_{i=1}^{4} P\{S_i\} = \alpha_1 \sum_{i=1}^{4} \prod_{j=2}^{4} P\{S_{i,j}\} \tag{7.1}$$

$$= \alpha_1\{(\alpha_2\beta_5)(\alpha_4\beta_8) + [(1\text{-}\alpha_2) + \alpha_2\beta_5](\alpha_3\beta_6)(\alpha_4\beta_9) + (\alpha_2\beta_5)(\alpha_3\beta_6)[(1\text{-}\alpha_4)$$

$$+ \alpha_4\beta_8](\alpha_4\beta_9) + (\alpha_2\beta_5)[(1-\alpha_3) + \alpha_3\beta_6](\alpha_3\beta_7)[(1-\alpha_4)+ \alpha_4\beta_8](\alpha_4\beta_9)\}$$

$$= \alpha_1\alpha_2\beta_5\alpha_4\beta_8 + \alpha_1\alpha_3\beta_6\alpha_4\beta_9\ (1-\alpha_2 + \alpha_2\beta_5) + \alpha_1\alpha_2\beta_5\alpha_3\beta_6\alpha_4\beta_9(1-\alpha_4 + \alpha_4\beta_8)$$

$$+ \alpha_1\alpha_2\beta_5\alpha_3\beta_7\alpha_4\beta_9(1-\alpha_3 + \alpha_3\beta_6)(1-\alpha_4 + \alpha_4\beta_8)$$

$$= \alpha_1\alpha_2\alpha_4\beta_5\beta_8 + \alpha_1\alpha_2\alpha_3\alpha_4\beta_5\beta_6\beta_9 + \alpha_1\alpha_3\alpha_4\beta_6\beta_9 - \alpha_1\alpha_2\alpha_3\alpha_4\beta_6\beta_9$$

$$+ \alpha_1\alpha_2\alpha_3\alpha_4\beta_5\beta_6\beta_8\beta_9 + \alpha_1\alpha_2\alpha_3\alpha_4\beta_5\beta_6\beta_7\beta_9(1-\alpha_4 + \alpha_4\beta_8)$$

$$= \alpha_1\alpha_2\alpha_4\beta_5\beta_8 + \alpha_1\alpha_3\alpha_4\beta_6\beta_9 + \alpha_1\alpha_2\alpha_3\alpha_4\beta_5\beta_6\beta_9 - \alpha_1\alpha_2\alpha_3\alpha_4\beta_6\beta_9$$

$$+ \alpha_1\alpha_2\alpha_3\alpha_4\beta_5\beta_6\beta_8\beta_9 + \alpha_1\alpha_2\alpha_3\alpha_4\beta_5\beta_6\beta_7\beta_8\beta_9$$

$$= \alpha_1\alpha_2\alpha_4\beta_5\beta_8 + \alpha_1\alpha_3\alpha_4\beta_6\beta_9 + \alpha_1\alpha_2\alpha_3\alpha_4\beta_6\beta_9(\beta_5 - 1 + \beta_5\beta_8 + \beta_5\beta_7\beta_8)$$

A number of analytical models have been proposed to address the problem of software reliability measurement. According to the nature of the failure process and based on the failure history of the software, these approaches can be classified as Time Between Failures (TBF) Models, Failure Count Model, Fault Seeding Models, and Input Domain Based Models [18]. The most common TBF model assumes that the time between the $(i\text{-}1)^{st}$ failure and $i^{th}$ failure independently follows a distribution whose parameters depend on the number of faults remaining in the program during the interval, embedded faults are independent and of equal probability of exposure, faults are removed immediately after each occurrence, no new faults are introduced during correction. Unlike in a regular manufacturing system, where hardware failure rate increases with time and maintenance, it is expected that the successive failure times will get longer as faults are removed from the node software system.

Since software fail only when they are executed, the calendar time doesn't represent the time during which the software could fail. The utilization of the software inside node $j$ --- $\varepsilon_j$ is used to compensate for the difference in the time domain.

We will analyze the reliability and availability of networks integrated with software failures and imperfect nodes based on MORIN [31], where the times between software failures follow the TBF models. The directed bridge network as shown in Figure 7.1 is used as the example. Hardware failures in each node are assumed to follow Poisson process with the same rate $\lambda_1$. Failure of each link also presumably follows the Poisson distribution with the same rate $\lambda_2$. Jelinski Moranda (JM) De-Eutrophication Model is adopted as the software failure model. The software in each node of the integrated network is assumed to have the same utilization $\varepsilon$ and follow the same stochastic failure process.

JM De-Eutrophication Model is one of the earliest and probably the most commonly used model for assessing software reliability. It assumes that there are $N$ software faults at the start of testing, and that each fault is independent of the others and equally likely to cause a failure during testing. A detected fault is removed with certainty in negligible time and no new faults are introduced during the debugging process. The software failure rate or the hazard function is assumed to be proportional to the current fault content of the program. It is expected that the successive failure times would become longer as faults are removed from the software system. Hence the hazard function during $t_i$, the time between the ($i$-1)st and $i$th failure, is given by

$h_s(t_i) = \Phi[N-(i-1)]\ \varepsilon$, where $\Phi$ is a proportionality constant, $\varepsilon$ is the software utilization coefficient.

Thus $R_s(t) = e^{-\int_0^t h_s(\delta)d\delta} = e^{-\Phi(N-i+1)\varepsilon t}$

$$f_s(t) = h_s(t)R_s(t) = h_s(t)e^{-\int_0^t h_s(\delta)d\delta} = \Phi(N-i+1)e^{-\phi(N-i+1)\varepsilon t}$$

In the bridge network, for the node software, based on the utilization $\varepsilon$, the operational probability is:

$$\alpha_{1s} = \alpha_{2s} = \alpha_{3s} = \alpha_{4s} = R_s(t) = e^{-\Phi(N-i+1)\varepsilon t}$$

For the node hardware, the operational probability is:

$$\alpha_{1h} = \alpha_{2h} = \alpha_{3h} = \alpha_{4h} = e^{-\lambda_1 t}$$

For the links, the operational probability is:

$$\beta_5 = \beta_6 = \beta_7 = \beta_8 = \beta_9 = e^{-\lambda_2 t}$$

The terminal reliability from $s$ to $t$ between the $(i-1)^{st}$ and $i^{th}$ software failure is thus:

$$R_{s\text{-}t} = \alpha_1\alpha_2\alpha_4\beta_5\beta_8 + \alpha_1\alpha_3\alpha_4\beta_6\beta_9 + \alpha_1\alpha_2\alpha_3\alpha_4\beta_6\beta_9(\beta_5 - 1 + \beta_5\beta_8 + \beta_5\beta_7\beta_8)$$

$$= \alpha_{1s}\alpha_{1h}\alpha_{2s}\alpha_{2h}\alpha_{4s}\alpha_{4h}\ \beta_5\beta_8 + \alpha_{1s}\alpha_{1h}\alpha_{3s}\alpha_{3h}\alpha_{4s}\alpha_{4h}\ \beta_6\beta_9$$

$$+ \alpha_{1s}\alpha_{1h}\alpha_{2s}\alpha_{2h}\alpha_{3s}\alpha_{3h}\alpha_{4s}\alpha_{4h}\ (\beta_5 + \beta_5\beta_8 + \beta_5\beta_7\beta_8 - 1)$$

$$= e^{-3\lambda_1 t}\, e^{-3\Phi(N-i+1)\varepsilon t}\, e^{-2\lambda_2 t} + e^{-3\lambda_1 t}\, e^{-3\Phi(N-i+1)\varepsilon t}\, e^{-2\lambda_2 t}$$

$$+ e^{-4\lambda_1 t}\, e^{-4\Phi(N-i+1)\varepsilon t}\, (e^{-\lambda_2 t} + e^{-2\lambda_2 t} + e^{-3\lambda_2 t} - 1)$$

$$= 2e^{-3\lambda_1 t}\, e^{-3\Phi(N-i+1)\varepsilon t}\, e^{-2\lambda_2 t} + e^{-4\lambda_1 t}\, e^{-4\Phi(N-i+1)\varepsilon t}\, (e^{-\lambda_2 t} + e^{-2\lambda_2 t} + e^{-3\lambda_2 t} - 1)$$

Denote $\theta = -[\lambda_1 + \Phi\varepsilon(N - i + 1)]t$, after the symbolic simplification,

$$R_{s\text{-}t} = 2e^{-3[\lambda_1+\phi\varepsilon(N-i-1)]t}\ e^{-2\lambda_2 t} + e^{-4[\lambda_1+\phi\varepsilon(N-i-1)]t}\left(e^{-\lambda_2 t} + e^{-2\lambda_2 t} + e^{-3\lambda_2 t} - 1\right)$$

$$= 2e^{3\theta}e^{-2\lambda_2 t} + e^{4\theta}e^{-\lambda_2 t} + e^{4\theta}e^{-2\lambda_2 t} + e^{4\theta}\ e^{-3\lambda_2 t} - e^{4\theta}$$

From the above symbolic expression, it can be concluded that the reliability of the studied network follows a multivariate distribution that is usually used to describe a system consisting of multiple components with different failure distributions. Furthermore, the network reliability depends on the software utilization, software failure rate and hardware failure rate inside a node, the failure rate of a link, and the total fault number in the software in each node.

### 7.1.2 Sample Network 2

Figure 7.3 shows the other sample network where only source node *s*, sink node *t*, and links are labeled.

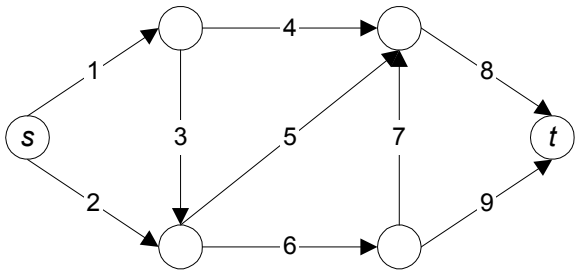



Figure 7.3 Sample Network 2

As illustrated in Figure 7.4, there are seven mutual exclusive successful events generated by MORIN method:

$S_1 = 148$ $S_2 = \bar{1}269$ $S_3 = \bar{1}26\bar{9}78$ $S_4 = \bar{1}2\bar{6}58$

$S_5 = 1\bar{4}369$ $S_6 = 1\bar{4}36\bar{9}78$ $S_7 = 1\bar{4}3\bar{6}58$

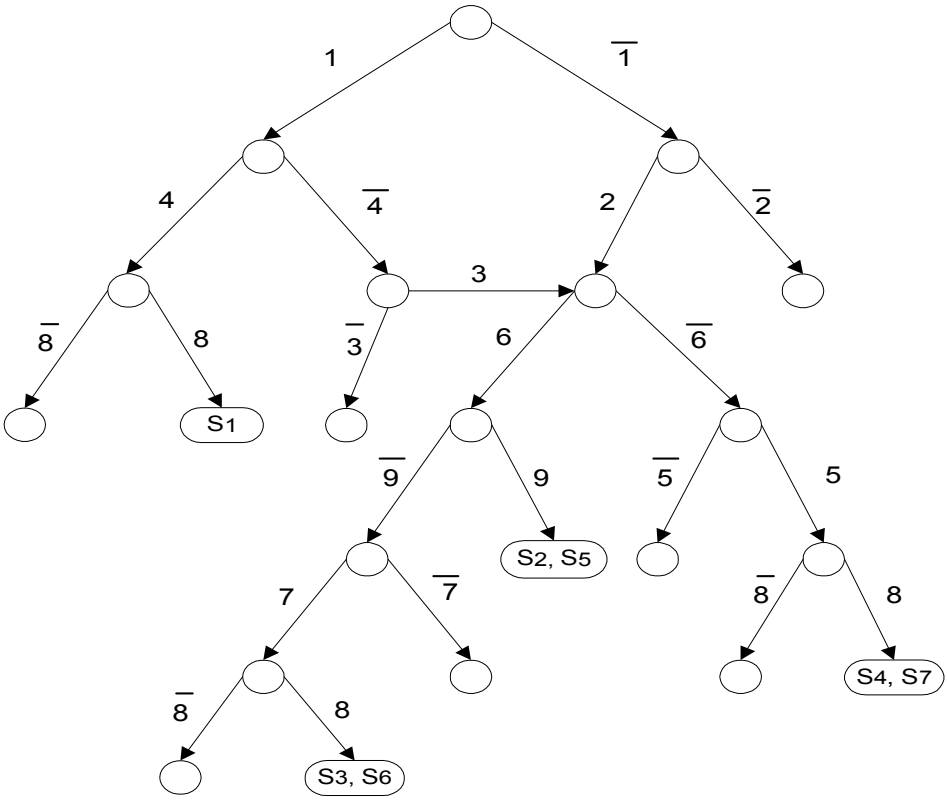



Figure 7.4. Event-Tree Generated by the MORIN Algorithm for Sample Network 2

Similarly as in Sample network 1, the network reliability can be calculated through the symbolic computations following the proposed MORIN method.

## 7.2 SAMOT Experiment Results[†]

To demonstrate the SAMOT tool, some experiments are conducted with following basic important assumptions.

- Operation, administration and maintenance (OA&M), as well as procedural errors, are not considered in the system and end-to-end availability modeling.
- The data path availability is not demonstrated in the experiments since typical data does not require real time transmission, the HA requirements are lower.
- Customer premium equipment (CPE) failures are not considered in the experiments. CPE is usually located on the customer side and is often mostly affected by non-product-quality-related failures in practice.
- Link failures are negligible in the experiments due to the extremely high reliability of links (fiber trunk or cooper cable).
- The end-to-end path does not include the Public Switch Telephone Network (PSTN) or other segments that the servers are connected to. In this sense, the end-to-end path is semi-end to end.
- To simplify the experiments, the operating system (OS) software and application software are integrated into a single software block in Redundancy Module (if not specified) albeit the OS software and application software usually fail with different distributions and should be considered separately when applicable.

[†] All experimental metrics showed in this section are intended as an illustration of the SAMOT tool only, and do not represent or imply actual reliability/availability configuration design and/or field performance of any product of any company.

### 7.2.1 Practical Networks

The architecture of a practical network (as in Figure 7.5) and the corresponding modeling flowchart are illustrated in Figure 7.6 and 7.7 respectively. Figure 7.8 shows the signaling path and bearer path transverses different component systems in the sample network.

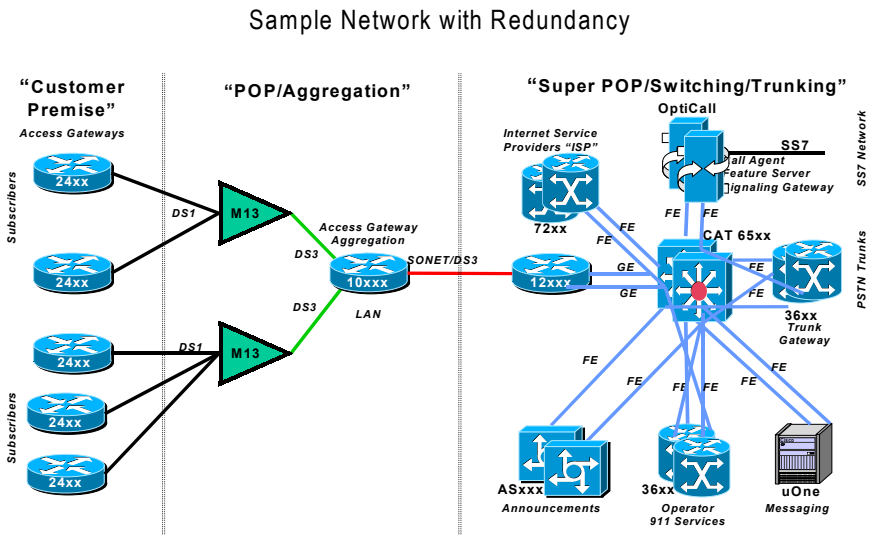



Figure 7.5. Architecture of A Sample Network with Redundancy

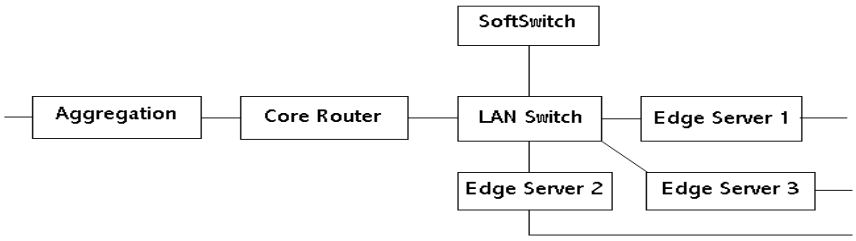



Figure 7.6. Block Diagram of A Sample Baseline Network

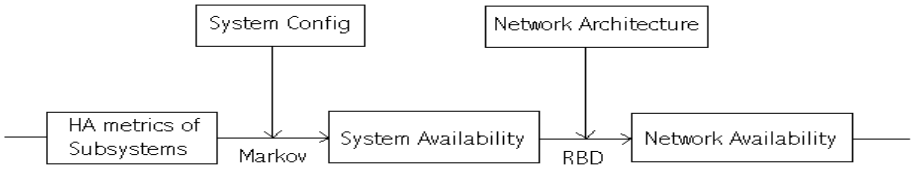



Figure 7.7. Modeling Flowchart for A Baseline Network

To improve the availability of the end-to-end path, while considering the cost factor, 1:1 box-level redundancy can be implemented in the critical SoftSwitch and less expensive LAN Switch and edge servers, as showed in Figure 7.8.

A dynamic protocol such as hot standby router protocol (HSRP) or ICMP router discovery protocol (IRDP) runs between the redundant SoftSwitches in order to quickly populate the routing table to the standby unit when a network failure occurs [120].

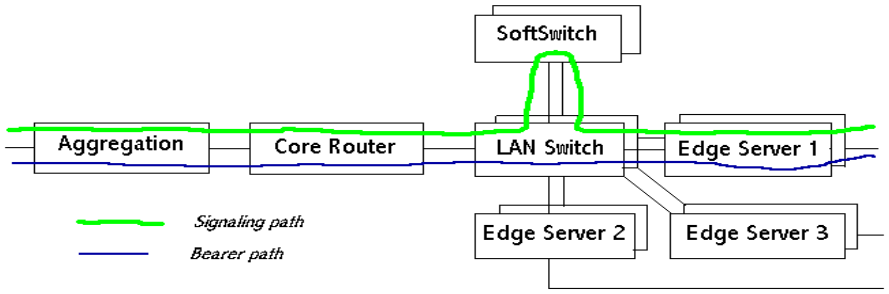



Figure 7.8. Block Diagram of A Sample Network with 1:1 System Redundancy

Figure 7.9 is the flowchart of modeling availability of a network with 1:1 redundancy.

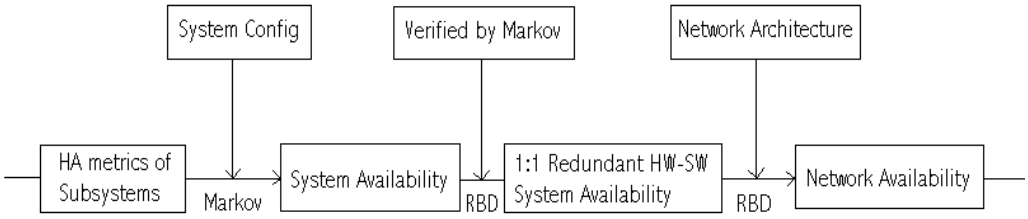



Figure 7.9. Modeling Flowchart for A Network with 1:1 System Redundancy

### 7.2.2 SAMOT Modeling Results

#### 7.2.2.1 System Availability

We first apply the SAMOT tool to calculate the availability metrics of each individual system based on its internal system configuration and subsystem reliability.

MTBF and MTTR of each subsystem are two basic availability parameters to compute the corresponding system availability. Switchover coverage and switchover time are another two important availability metrics if redundancies are involved. The first two "hours" columns in Table 7.1-7.5 are inputs of the SAMOT tool in order to compute the system availability and end-to-end network availability. MTBF is calculated according to the Bellcore standards, MTTR is estimated based on the system HA configurations and features as well as part staffing condition. The last four columns (from right of the table) are system availability metrics output from SAMOT.

Table 7.1. Availability Metrics of Aggregation Device

| Component Description | MTBF (hr) | MTTR (hr) | Annual Downtime (min) | A (%) | DPM (B) | DPM (D) |
|---|---|---|---|---|---|---|
| Aggre. Dev Chassis | 674,310 | 4 | 3.235 | 99.9994 | 6.15 | 0.15 |
| Processor, with 1:1 R | 128,152 | 2 | 1.167 | 99.9998 | 2.22 | 0.43 |
| CT3 Card | 230,886 | 2 | 4.619 | 99.99912 | 8.79 | 0.47 |
| COC12 Card | 172,604 | 2 | 6.156 | 99.99883 | 11.71 | 0.54 |
| Power, 1:1 load-sharing redundancy | 158,228 | 2 | 0.143 | 99.99997 | 0.27 | 0.013 |
| OS Software | 33,835 | 0.058 | 0.906 | 99.99983 | 1.724 | 1.478 |
| SW upgrade | 4,380 | 0.058 | 9.599 | 99.99817 | 18.26 | 17.12 |
| Total Aggre. Device | 61,097 | 3 | 25.825 | 99.99509 | 49.13 | 20.20 |

Note: DPM(B) is the DPM for blocked calls and DPM(D) is the DPM for dropped calls.

Table 7.2. Availability Metrics of Core Router

| Component Description | MTBF (hr) | MTTR (hr) | Annual Downtime (min) | A (%) | DPM (B) | DPM (D) |
|---|---|---|---|---|---|---|
| Core Router Chassis | 297,137 | 4 | 7.518 | 99.99857 | 14.30 | 0.336 |
| Processor, w/ 1:1 R | 108,304 | 2 | 2.283 | 99.99957 | 4.344 | 0.512 |
| Feature Card | 272,584 | 2 | 0.077 | 99.99999 | 0.147 | 0.004 |
| Feature Card | 422,115 | 2 | 0.050 | 99.99999 | 0.095 | 0.002 |
| Alarm Card | 845,123 | 2 | 1.244 | 99.99976 | 2.366 | 0.059 |
| 4OC3 Card | 164,046 | 2 | 6.947 | 99.99868 | 13.22 | 1.783 |
| 4OC12 Card | 124,440 | 2 | 8.987 | 99.99829 | 17.10 | 1.880 |
| Power, 1:1 load-sharing redundancy | 316,456 | 2 | 0.748 | 99.99999 | 0.142 | 0.006 |
| OS Software | 33,835 | 0.251 | 3.905 | 99.99926 | 7.430 | 1.478 |
| SW upgrade | 4,380 | 0.251 | 45.123 | 99.99142 | 85.85 | 17.12 |
| Total Core Router | 20,687 | 3 | 76.208 | 99.98550 | 145.0 | 23.18 |

Table 7.3. Availability Metrics of SoftSwitch

| Component Description | MTBF (hr) | MTTR (hr) | Annual Downtime (min) | A (%) | DPM (B) | DPM (D) |
|---|---|---|---|---|---|---|
| E-Switch HW, 1:1 box Redundancy | 164,528 | 2 | 0.776 | 99.99854 | 1.458 | 0.099 |
| E-Switch IOS-R | 18,039 | 0.108 | 0.111 | 99.99998 | 0.211 | 0.302 |
| Fru Server (1:1 R) | 51,810 | 2 | 2.304 | 99.99956 | 4.384 | 0.210 |
| SoftSwitch Software | 22,545 | 0.083 | 0.060 | 99.99999 | 0.114 | 0.302 |
| SW upgrade | 4,380 | 0.083 | 0.458 | 99.99991 | 0.871 | 1.244 |
| Total SoftSwitch | 428,568 | 3 | 3.700 | 99.99930 | 7.039 | 2.158 |

Note: Power is not considered in this SoftSwitch model due to using the Central Office power.

Table 7.4. Availability Metrics of LAN Switch

| Component Description | MTBF (hr) | MTTR (hr) | Annual Downtime (min) | A (%) | DPM (B) | DPM (D) |
|---|---|---|---|---|---|---|
| LAN Switch Chassis | 369,897 | 4 | 6.039 | 99.99885 | 11.49 | 0.270 |
| Processor Engine 1:1R | 41,988 | 2 | 3.825 | 99.99927 | 7.277 | 0.485 |
| Switch Fabric Mod. | 172,889 | 2 | 0.826 | 99.99984 | 1.571 | 0.071 |
| OS Software | 18,039 | 0.058 | 0.185 | 99.99996 | 0.353 | 0.302 |
| Application Software | 18,039 | 0.058 | 0.185 | 99.99996 | 0.353 | 0.302 |
| SW Upgrade | 4,380 | 0.367 | 1.925 | 99.99963 | 3.663 | 1.244 |
| Power, w/ 1:1 Load-Sharing R | 316,456 | 2 | 0.075 | 99.99999 | 0.142 | 0.006 |
| Line Card | 93,457 | 2 | 12.947 | 99.99754 | 24.63 | 3.307 |
| Connector | 94,684 | 2 | 12.802 | 99.99756 | 24.36 | 3.299 |
| 9 Slot Fan w/ 1:1 Load Sharing R | 740,740 | 2 | 0.028 | 99.99999 | 0.054 | 0.001 |
| Total LAN Switch | 40,592 | 3 | 38.837 | 99.99261 | 73.89 | 9.289 |

Table 7.5. Availability Metrics of Edge Server 1

| Component Description | MTBF (hr) | MTTR (hr) | Annual Downtime (min) | A (%) | DPM (B) | DPM (D) |
|---|---|---|---|---|---|---|
| Server1 Chassis | 45,212 | 3 | 37.780 | 99.99281 | 71.88 | 2.212 |
| DSP Module | 594,126 | 2 | 3.430 | 99.99935 | 6.526 | 4.824 |
| DMM Modem with Feature Card | 63,404 | 2 | 18.240 | 99.99653 | 34.70 | 5.528 |
| OS Software | 10,549 | 0.192 | 5.232 | 99.99901 | 9.953 | 4.740 |
| Software Upgrade | 4,380 | 0.350 | 29.399 | 99.99441 | 55.93 | 11.42 |
| Power, with 1:1 Load Sharing R | 600,000 | 2 | 1.986 | 99.99962 | 3.778 | 0.250 |
| Total Edge Server 1 | 16,408 | 3 | 96.066 | 99.98172 | 182.8 | 28.97 |

Further details of the model can be referred to Appendices.

#### 7.2.2.2 Availability of 1:1 Redundant Systems

Inside a system box, it is difficult to deploy redundancy on the ingress card and egress card to eliminate the single points of failure (SPF); the system chassis is always a SPF. The effect of SPFs usually accumulates to be the bottleneck of achieving the carrier class (five 9s) network availability. Thus to better improve the overall end-to-end availability per customer's HA requirements, 1:1 active/standby redundancies at the box-level is usually suggested to some critical systems or inexpensive systems in addition to board-level redundancy for key components in the system. SAMOT can accurately model the availability of a complex hardware-software system with redundancy schemes.

Since a Markov model is capable of exhaustively enumerating the failure states and their transitions, it is used here to verify the correctness and accuracy of the SAMOT tool for calculating the availability of a 1:1 redundant system. The Bellcore Systems Reliability

Analysis Software (SRAS) Ver 2.2 (referring to Appendix) is used as the Markov modeling tool in this chapter.

Table 7.6. Comparisons of Availability Modeling Results on Unplanned Outages of 1:1 Redundant System by SAMOT and Markov

| ***Systems*** | | A(Soft-S) (%) | A(LAN-S) (%) | A(Edge.) (%) |
|---|---|---|---|---|
| Case 1: ASC = 0.99 SSC = 0.90 ST = 10 sec | SAMOT | 99.999907 | 99.999316 | 99.998802 |
| | Markov | 99.999909 | 99.999332 | 99.998830 |
| | Discrepancy | 0.000002 | 0.000016 | 0.000028 |
| Case 3: ASC = 0.99 SSC = 0.90 ST = 30 sec | SAMOT | 99.999878 | 99.999278 | 99.998737 |
| | Markov | 99.999880 | 99.999294 | 99.998764 |
| | Discrepancy | 0.000002 | 0.000016 | 0.000027 |
| Case 3: ASC = 0.99 SSC = 0.80 ST = 10 sec | SAMOT | 99.999832 | 99.998676 | 99.997679 |
| | Markov | 99.999852 | 99.998847 | 99.997979 |
| | Discrepancy | 0.000020 | 0.000171 | 0.000300 |
| Case 4: ASC = 0.99 SSC = 0.80 ST = 30 sec | SAMOT | 99.999807 | 99.998642 | 99.997621 |
| | Markov | 99.999826 | 99.998812 | 99.997919 |
| | Discrepancy | 0.000019 | 0.000170 | 0.000298 |
| Case 5: ASC = 0.90 SSC = 0.80 ST = 10 sec | SAMOT | 99.999821 | 99.998597 | 99.997541 |
| | Markov | 99.999796 | 99.998361 | 99.997129 |
| | Discrepancy | 0.000025 | 0.000236 | 0.000412 |
| Case 6: ASC = 0.90 SSC = 0.80 ST = 30 sec | SAMOT | 99.999798 | 99.998566 | 99.997488 |
| | Markov | 99.999772 | 99.998330 | 99.997074 |
| | Discrepancy | 0.000026 | 0.000236 | 0.000414 |
| Case 7: ASC = 0.90 SSC = 0.70 ST = 10 sec | SAMOT | 99.999754 | 99.998014 | 99.996520 |
| | Markov | 99.999752 | 99.997988 | 99.996475 |
| | Discrepancy | 0.000002 | 0.000026 | 0.000045 |
| Case 8: ASC = 0.90 SSC = 0.70 ST = 30 sec | SAMOT | 99.999734 | 99.997988 | 99.996474 |
| | Markov | 99.999730 | 99.997959 | 99.996425 |
| | Discrepancy | 0.000004 | 0.000029 | 0.000049 |

Note: 1. Denote: ASC/SSC –Active/Standby Switchover Coverage, ST-Switchover Time

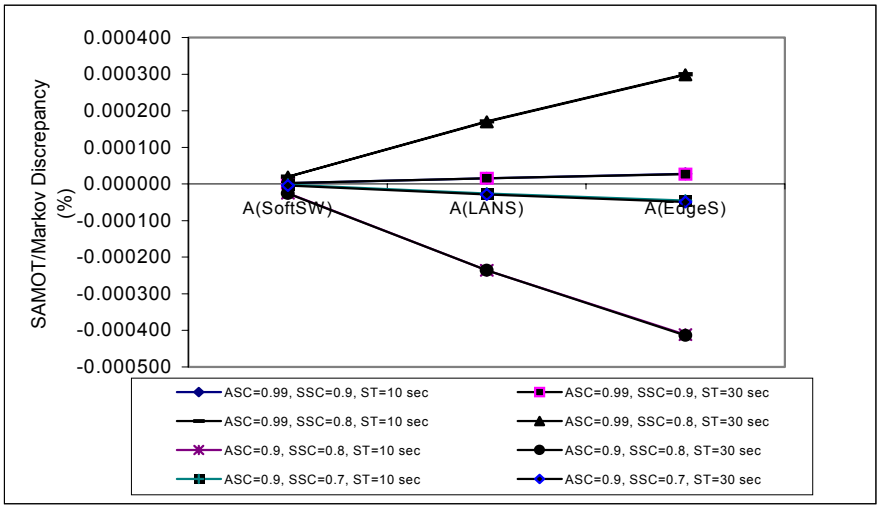
2. The MTBF numbers for unplanned hardware outage of Soft-S, LAN-S, and Edge are respectively 513522, 47574, and 27130 hours.

3. A(Edge.)(%) is the availability of unplanned outage of Edge Server1

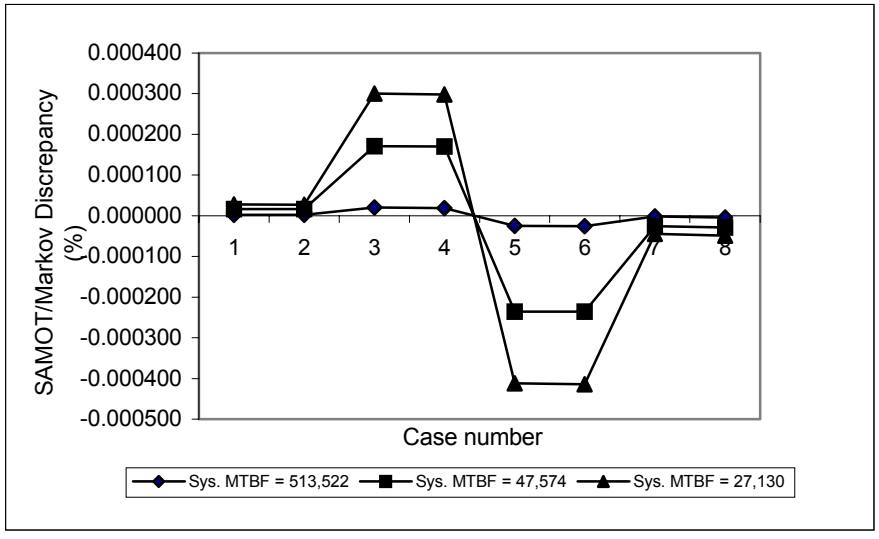
Results in Table 7.6 indicate that the availability value for a 1:1 redundant hardware-software system derived by the SAMOT tool is extremely accurate, comparing to the

Markov analysis results. Under the above experimental parameter sets, SAMOT just has a discrepancy from 0.000002% to 0.00045%.

Sensitivity analysis of the modeling results in Figure 7.10(a) shows that there is little difference of results among different switchover time (10 seconds and 30 seconds) and only 4 lines are visible, therefore the switchover time does not seem to be a significant factor affecting SAMOT's accuracy.




(a)




(b)

Figures 7.10(a ) & (b). Discrepancy of SAMOT & Markov Modeling Results

Figure 7.10(b) shows that the higher the switchover coverage is, the more accurate the SAMOT will be; SAMOT accuracy becomes more sensitive to the switchover coverage when the studied system is less reliable (i.e., with a lower MTBF).

#### 7.2.2.3 Network Path Availability

Table 7.7 is the availability metrics of the paths in the sample network based on the above network architecture, system configuration and subsystem availability parameters.

Table 7.7. Availability of Signaling Path and Bearer Path of the Sample Network

| Network Path | Annual Downtime (min) | A (%) | DPM (B) | DPM (D) |
|---|---|---|---|---|
| Signaling Path | 116.15 | 99.9779 | 220.98 | |
| Bearer Path | 115.47 | 99.9780 | | 76.60 |

Note: The above results are based on Case 1 parameter settings.

In general, the SAMOT tool is very accurate when applied on availability modeling and analysis for a network comprised of redundant systems with high switchover coverage and high system availability. The switchover time between the active and standby systems does not seem to be a very significant factor affecting the SAMOT accuracy.

# CHAPTER 8

# CONCLUSIONS AND FUTURE RESEARCH

This dissertation aims to develop efficient approaches to analyze the reliability and availability of networks integrated with link failures, node hardware failure and software failures. The research methodologies and results are performed at the system level and the network level. It will be the author's great pleasure that this research has added some valuable contributions in the network reliability and availability field:

- An efficient approach - MORIN is proposed and demonstrated.
- A simplified methodology and modeling tool for solution availability - SAMOT is developed and illustrated for modeling the end-to-end availability of a network comprised of 1:1 redundant hardware-software systems. SAMOT requires the network architecture, system configurations, the MTBF, MTTR of subsystems of each system along the path and the redundancy availability parameters as inputs. SAMOT results are verified by Markov analysis and can be validated by field collected availability data.
- Petri nets based techniques and efficient modeling tools for parallel and concurrent systems are discussed and explored as well.

The major object of the research is *s-t* two terminal reliability and availability problems. MORIN can identify the event trees and find the path and calculate the overall network reliability, but short of capturing the scenarios when redundancies are involved in complex component systems (nodes) that are subject to software and hardware failures. On the other side, the SAMOT models the reliability and availability of complex systems, and can also compute the end-to-end solution availability, given the network architecture and solution path. The SAMOT Main Module can provide reliability of component system to Event_RCal Module of MORIN.

MORIN and SAMOT are very well complementary approaches that integrate into a comprehensive solution package for modeling the reliability and availability of complex networks. As illustrated in Figure 8.1, the package addressing the practical problems comprises of two segments: the proposed MORIN firstly identifies the disjointed event trees and path sets from source node *s* to sink node *t*; then the SAMOT is developed to solve the path set problem by computing and approximating (with high accuracy) the reliability and availability of practical end-to-end solutions consisting of integrated hardware-software systems (with redundancies).

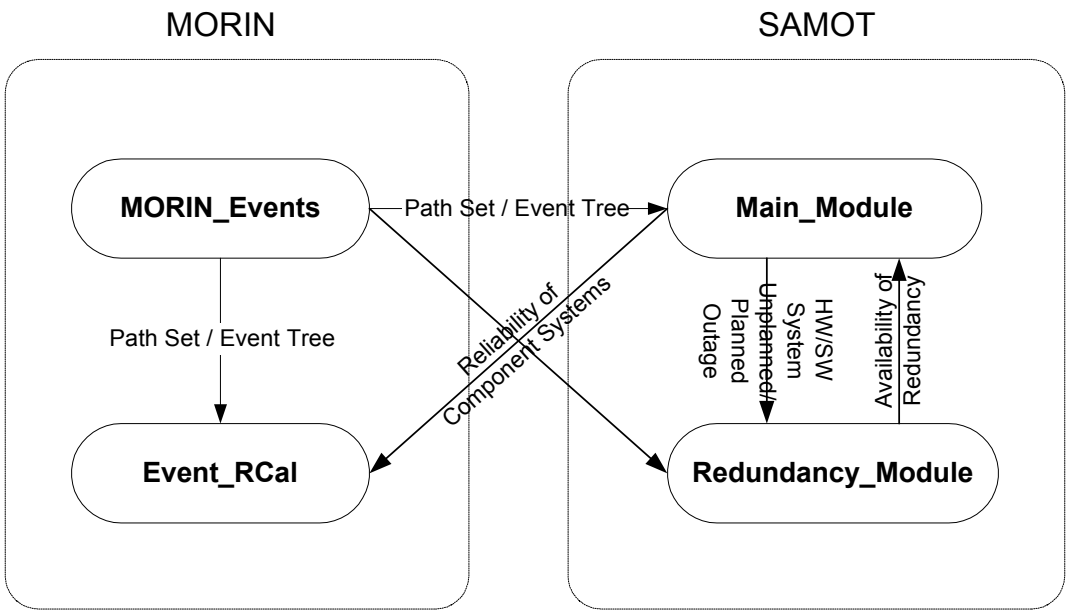



Figure 8.1 Complementary Relationship Between MORIN and SAMOT

Follow-up researches can be logically expanded to analyzing the network reliability of *k*-terminals and all-terminals. Future researches in reliability and availability analysis for integrated networks can also address the different impact on the failure of its incidental node from each (category of) software fault. Some extended models would be developed based on empirical software failure data. Another research direction is the study of the dependency of software failures and hardware failures that cause node failure.

It would be a very rewarding task to extend the SAMOT application to the end-to-end path availability of a network with 1:N software-hardware system redundancy. Finally, should more resource and efforts be available in applying the special programming language and relevant software package, the sketchy PN-based methodologies would have been better developed and verified.

## REFERENCES


1. R.D. Shier, *Network Reliability and Algebraic Structures*, Clarendon Press, Oxford, 1991.

2. K.K. Aggarwal, J.S. Gupta, K.B. Misra, A Simple Method for Reliability Evaluation of a Communication System, *IEEE Trans. Communications*, May 1975, pp563-566.

3. K.K. Aggarwal, K.B. Misra, J.S. Gupta, A Fast Algorithm for Reliability Evaluation, *IEEE Trans. Reliability*, Vol.R-24, No.1, April 1975, pp83-85.

4. O.M. Ball, Computational Complexity of network Reliability Analysis: An Overview, *IEEE Trans. Reliability*, Vol.R-35, No.3, August 1986, pp230-239

5. K. Sutner, A. Satyanarayana, C. Suffel, The Complexity of the Residual Node Connectedness Reliability Problem, *SIAM Journal of Computing*, Vol.20, No.1, February 1991, pp.149-155.

6. W.J. Ke, S.D. Wang, Reliability Evaluation for Distributed Computing Networks with Imperfect Nodes, *IEEE Trans. Reliability*, Vol.R-46, No.1, September 1997, pp342-349.

7. D. Torrieri, Calculation of Node-pair Reliability in Large Networks with Unreliable Nodes, *IEEE Trans. Reliability*, Vol.R-43, No.3, September 1994, pp375-377.

8. K.B. Misra, T.S.M. Rao, Reliability Analysis of Redundant Networks Using Flow Graphs, *IEEE Trans. Reliability*, Vol.R-19, February 1970, pp19-24.

9. Y.H. Kim, K.E. Case, P.M. Ghare, A Method for Computing Complex System Reliability, *IEEE Trans. Reliability*, Vol.R-21, November 1972, pp215-219.

10. K.K. Aggarwal, J.S. Gupta, K.B. Misra, A New Method for System Reliability Evaluation, *Microelectronic Reliability*, Vol.12, No.5, November 1973, pp435-440.

11. W. Everett, S. Keene, A. Nikora, Applying Software Reliability Engineering in the 1990s, *IEEE Trans. Reliability*, Vol.47, No.3-SP, September 1998, pp372SP -378SP.

12. M. Lipow, On Software Reliability: A Preface by the Guest Editor, *IEEE Trans. Reliability*, Vol.R-28, No.3, August 1979.

13. V.A. Nets, B.P. Filin, Consideration of Node Failures in the Network Reliability Calculation*, IEEE Trans. Reliability*, Vol.45, March 1996, pp127-128.

14. V.K.P. Kumar, S. Hariri, C.S. Raghavendra, Distributed Program Reliability Analysis, *IEEE Trans. Software Engineering*, Vol.SE-12, January 1986, pp42-50.

15. Y.B. Yoo, N. Deo, A Comparison of Algorithms for Terminal pair Reliability, *IEEE Trans. Reliability*, Vol. 37, June 1988, pp210-215.

16. S. Rai, A. Kumar, and E.V. Prasad, Computing Terminal Reliability of Computer Networks, *Reliability Engineering*, Vol. 16, 1986, pp109-119.

17. C.J. Colbourn, *the Combinatorics of Network Reliability*, Oxford University Press, New York, 1987.

18. R. Bhandari, *Survivable Networks, Algorithms for Diverse Routing*, Kluwer Academic Publishers, 1999.

19. E.F. Moore, C.E. Shannon, Reliable Circuits Using Less Reliable Relays, *Journal of the Franklin Institute*, Vol. 262, 191-208, 281-297.

20. L.R. Jorge, A.D. Kieron, Classifying Combined Hardware/Software R Models, *Proceedings of Annual Reliability and Maintainability Symposium*, 1984, pp282-288.

21. A.L. Goel, Software Reliability Models: Assumption, Limitations, and Applicability, *IEEE Trans. Software Engineering*, Vol. SE-11, No.12, December 1985, pp1411-1423.

22. J.B. Bowles, V. Swaminathan, A Combined Hardware, Software and Usage Model of Network Reliability and Availability, *IEEE 9th Annual International Phoenix Conference on Computers and Communications*, 1990, pp649-654.

23. F.T. Boesch, Synthesis of Reliable Networks – A Survey, *IEEE Trans. Reliability*, Vol. 35, August 1986, pp240-246.

24. A. Rosenthal, A Computer Scientist looks at Reliability Computations, *SIAM J. Computing*, 1975, pp133-152.

25. L.G. Valiant, The Complexity of Enumeration and Reliability Problems, *SIAM J. Computing*, Vol. 8, 1979, pp410-421.

26. J.S. Provan, M.O. Ball, The Complexity of Counting Cuts and Computing the Probability that a Graph is Connected, *SIAM J. Computing*, Vol. 12, 1983, pp777-788.

27. K.B. Misra, An Algorithm for Reliability Evaluation of Redundant Networks, *IEEE Trans. Reliability*, Vo. R-19, November 1970, pp146-151.

28. E.V. Krishnamurphy, G. Komissar, Computer-aid Reliability Analysis of Complicated Networks, *IEEE Trans. Reliability*, Vol. R-21, May 1972, pp86-89.

29. E. Hansler, A Procedure for Calculating the Reliability of a Communication Network, *Arch. Elek. Ubertragung*, Vol. 25, 1971, pp573-575.

30. R.B. Hurley, Probability maps, *IEEE Trans. Reliability*, Vol. R-12, September 1963, pp39-44.

31. W. Hou, O.G. Okogbaa, Reliability Analysis for Integrated Networks with Unreliable Nodes and Software Failures in the Time Domain, *Proceedings of Annual Reliability and Maintainability Symposium*, 2000, pp113-117.

32. K.K. Vemuri, J.B. Dugan, Reliability Analysis of Complex Hardware-Software Systems, *Proceedings of Annual Reliability and maintainability Symposium,* 1999, pp178-182.

33. E. Froncrak, A Top-down Approach to High-Consequence Failure Analysis for Software Systems, *ISSRE*, November 1997.

34. A. Satyanarayana, A. Prabhakar, New Topological Formula and Rapid Algorithm for Reliability Analysis of Complex Networks, *IEEE Trans. Reliability*, Vol. R-27, 1978, pp82-100.

35. F.T. Boesch, A. Satyanarayana, and C.L. Suffel, Some Alternate Characterizations of Reliability Domination, *Probability in the Engineering and Informational Science*, Vol. 4, 1990, 257-76.

36. M.O. Locks, Recursive Disjoint Products: A Review of Three Algorithms, *IEEE Trans. Reliability*, Vol. R-31, 1982, pp33-35.

37. M.O. Locks, A Minimizing Algorithm for Sum of Disjoint Products, *IEEE Trans. Reliability*, Vol. R-36, 1987, pp445-453.

38. H. Nakazawa, Bayesian Decomposition Method for Computing the Reliability of an Oriented Network, *IEEE Trans. Reliability*, Vol. R-25, 1976, pp77-80.

39. M.O. Ball, E.P. Cameron, Experiments with Network Reliability Analysis Algorithms, *Proceedings of the 17th Annual Conference on Modeling and Simulation*, Pittsburgh, 1986, pp1799-1803.

40. L.B. Page, J.E. Perry, Reliability of Directed Networks Using the Factoring Theorem, *IEEE Trans. Reliability*, Vol. R-38, 1989, pp556-562.
41. R. Johnson, Network Reliability and Acyclic Orientations, *Networks*, Vol. 14, 1984, pp489-505.

42. R.K. Wood, Factoring Algorithms for Computing K-terminal Network Reliability, *IEEE Trans. Reliability*, Vol. R-35, 1986, pp269-278.

43. H. Frank, Maximally Survival Node Vulnerable Networks, *Memorandum for File, Div. Emergency preparedness of Office of the President*, Washington D.C., March 1969.

44. H. Frank, Maximally Reliable Node Weighted Graphs*, Proceedings 3rd Annual Conference Information Sciences and Systems*, May 1969, pp1-6.

45. H. Frank, Some New Results in the Design of Survivable Networks*, Proceedings of 12th Annual Midwest Circuit Theory Symposium*, September 1969, ppI3.1-I3.8.

46. C. Colbourn, A. Satyanarayana, C. Suffel, K. Sutner, Computing the Residual Node Connectedness Reliability Problem, *SIAM J. Computing*, Vol. 20, 1991, pp149-155.

47. C. Colbourn, A. Satyanarayana, C. Suffel, On Residual Connectedness Network Reliability*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, Vol. 5, 1991, pp51-59.

48. C. Suffel, C. Stivaros, Uniformly Optimal networks in the Residual Node Connectedness Reliability Model, *Congressus Numerantium*, Vol. 81, March 1991, pp51-64.

49. F.T. Boesch, X. Li, C. Suffel, On the Exsitence of Uniformly Optimally Reliable Networks, *Networks*, Vol. 21, 1994, pp181-194.

50. O. Goldschmidt, P. Jaillet, R. LaSota, On Reliability of Graphs with Node Failures, *Networks*, Vol. 24, 1994, pp251-259.

51. W. Myrvold, K. Cheung, L. Page, J. Perry, Uniformly Most Reliable Graphs Do Not Always Exist, *Networks*, Vol. 21, 1991, pp417-419.

52. A. Amin, K. Siegrist, P. Slater, On the Nonexistence if Uniformly Optimal Graphs for Pair-connected Reliability, *Networks*, Vol. 21, 1991, pp359-368.

53. A. Amin, K. Siegrist, P. Slater, On Uniformly Optimally Reliable Graphs for Pair-connected Reliability with Vertex Failures, *Networks*, Vol. 23, 1993, pp185-193.

54. H.A. Fotoh, C. Colbourn, Computing 2-terminal Reliability for Radio-broadcast Networks, *IEEE Trans. Reliability*, Vol. R-38, December 1989, pp538-555.

55. H.A. Fotoh, C. Colbourn, Efficient Algorithms for Computing the Reliability of Permutation and Interval Graphs, *Networks*, Vol. 20, 1990, pp883-898.

56. J. Reynolds, *the Craft of Programming*, Englewood Cliffs, NJ, Prentice Hall, 1981.

57. S. Gerhart, L. Yelowitz, Observations of Fallibility in Applications of Modern Programming Methodologies, *IEEE Trans. Software Engineering*, Vol. SE-2, May 1976, pp195-207.

58. P.B. Moranda, Prediction of Software Reliability During Debugging, *Proceedings of Annual Reliability and Maintenance Symposium*, Washington DC, January 1975, pp327-332.

59. G.J. Schick, R.W. Wolverton, An Analysis of Computing Software Reliability Model, *IEEE Trans. Software Engineering*, Vol. SE-4, 1978, pp104-120.

60. A.L. Goel, K. Okumoto, An Analysis of Recurrent Software Failures in a Real-time Control System, *Proceedings of ACM Annual Technology Conference*, Washington DC, 1978, pp496-500.

61. A.L. Goel, K. Okumoto, A Markovian Model for Reliability and Other Performance Measures of Software Systems, *Proceedings of National Computing Conference*, New York, Vol. 48, 1979, pp769-774.

62. B. Littlewood, J.L. Verrall, A Bayesian Reliability Growth Model for Computer Software, Application Statistics, Vol. 22, 1973, pp332-346.

63. B. Littlewood, Theories of Software Reliability: How Good Are They and How Can They Be Improved? *IEEE Trans. Software Engineering*, Vol. SE-6, 1980, pp489-500.

64. A.L. Goel, K. Okumoto, A Time Dependent Error Detection rate Model for Software Reliability and Other Performance Measures, *IEEE Trans. Reliability*, Vol. R-28, 1979, pp206-211.

65. A.L. Goel, *A Guidebook for Software Reliability Assessment*, Rep. RADC-TR-83-176, August 1982.

66. A.L. Goel, *Software Reliability Modeling and Estimation Techniques*, Rep. RADC-TR-82-263, October 1982.

67. J.D. Musa, A Theory of Software Reliability and Its Application, *IEEE Trans. Software Engineering*, Vol. SE-1, 1971, pp312-327.

68. W.D. Brooks, R.W. Motley, *Analysis of Discrete Software Reliability Models*, Rep. RADC-TR-80-84, April 1980.

69. H.D. Mills, *On the Statistical Validation of Computer Programs,* IBM Federal System Division, Geithersburg, MD. 1975, Rep.72-6015.

70. M. Lipow, Estimation of Software packet Residual Errors, TRW, Redondo Beach, CA, 1972, *Software Series Rep. TRW_SS-72-09*.

71. S.L. Basin, Estimation of Software Error Rate Via Capture-recapture Sampling, *Science Applications Inc.,* Palo, Alto, CA, 1974.

72. E. Nelson, Estimating Software Reliability from Test Data*, Microelectronic Reliability*, Vol. 17, 1978, pp67-74.

73. C.V. Ramamoorthy, F.B. Bastani, Software Reliability: Status and Perspectives, *IEEE Trans. Software Engineering*, Vol. SE-8, July 1982, pp359-371.

74. M.R. Garey, D.S. Johnson, *Computers and Intractability, A Guide to the Theory of NP-Completeness*, W.H. Freeman and Company, 1979.

75. D.P. Siewierek, R.S. Swarz, *Reliable Computer Systems Design and Evaluation*, 3rd edition, A K Peters Ltd., 1998.

76. L.G. Valiant, the Complexity of Computing the Permanent*, Theoretical Computer Science*, Vol. 8, 1979, pp189-201.

77. L.G. Valiant, the Complexity of Enumeration and Reliability Problems*, SIAM J. Computing*, Vol. 8, 1979, pp410-421.

78. U. Sumita, Y.Masuda, Analysis of Software Availability/Reliability Under the Influence of Hardware Failures, *IEEE Trans. On Software Engineering*, Vol.SE-12, No.1, 1986, pp32-41.

79. A.L. Geol, J. Soenjoto, Models for Hardware-Software System Operational-performance Evaluation, *IEEE Trans. Reliability*, Vol.R-31, No.3, 1981, pp232-239.

80. J.E. Angus, L.E. James, Combined Hardware/Software Reliability Models*, Proc. Annual Reliability and Maintainability Symposium*, 1982, pp176-181.

81. B. Cappelle, E.E. Kerre, Issues in Possibilistic Reliability Theory*, Reliability and Safety Analyses under Fuzziness*, Physica-Verlag, 1995, pp61-80.

82. H. Tanaka, L.T. Fan, F.S. Lai, K. Toguchi, Fault Tree Analysis by Fuzzy Probability, *IEEE Trans. Reliability*, Vol.32, 1983, pp453-457.

83. D. Singer, A Fuzzy Set Approach to Fault Tree and Reliability Analysis*, Fuzzy Sets and Systems*, Vol.34, 1990, pp145-155.

84. K. Cai, C. Wen, Street-lighting Lamp Replacement: a Fuzzy Viewpoint, *Fuzzy Sets and Systems* , Vol.37, 1990, pp161-172.

85. M.A. Marsan, *et. al.*, Introduction to Generalized Stochastic Petri Nets, *Microelectronic Reliability,* v 31 n 4 1991 p 699-725.

86. M.A. Marsan, *et. al.*, On Petri Nets with Stochastic timing, *International Workshop on Time Petri Nets*, IEEE Computer Society Press, 1985, pp80-87.

87. M.A. Holliday, M.K. Vernon, A Generalized Timed Petri Net Model for Performance Analysis, *International Workshop on Time Petri Nets*, IEEE Computer Society Press, 1985, pp180-190.

88. O. Botti, F. De Cindio, Process and Resource Boxes: An Integrated PN Performance Model for Applications and Architectures, *IEEE Proc. of the International Conference on Systems, Man and Cybernetics*, Le Toquet, France, 1993.

89. S. Donetelli, G. Franceschinis, The PRS methodology: Integrating Hardware and Software Models, *Lecture notes in Computer Science*, Springer, 1997, pp133-151.

90. W. Reisig, *Pertri Nets, An Introduction*, Springer-Verlag, 1982.

91. W. Reisig, G. Rozenberg*, Lectures on Petri Nets I: Basic Models, Advances in Petri Nets*, Springer-Verlag, 1998.

92. W. Reisig, G. Rozenberg*, Lectures on Petri Nets I: Applications, Advances in Petri Nets*, Springer-Verlag, 1998.

93. M.A. Marsan, G. Balbo, K. Trivedi, *International Workshop on Time Petri Nets*, IEEE Computer Society Press, 1985.

94. K. Jensen, *Coloured Petri Nets, Basic Concepts, Analysis Methods and Practical Use*, Volume 1, 2nd Edition, Springer-Verlag, 1996.

95. M. Balakrishnan, Stochastic Petri Nets for the Reliability Analysis of Communication Network Applications with Alternate-routing, *Reliability Engineering & System Safety,* Vol.52, n.3 Jun 1996, pp 243-259.

96. S.M. Koriem, Fault-tolerance Analysis of Hypercube Systems Using Petri Net Theory, *Journal of Systems and Software*, Vol.21, n.1, April 1993. pp 71-88.

97. W.G. Schneeweiss, *Petri Nets for Reliability Modeling* (*in the Fields of Engineering Safety and Dependability*), LiLoLe-Verlag GmbH (Publishing Co. Ltd), 1999.

98. A.D. Stefano, O. Mirabella, Evaluating the Fieldbus Data Link Layer by a Petri Net-based Simulation, , *IEEE Trans. Industrial Electronics*, Vol.38, No.4, August 1991.

99. G. Juanole, Y. Atamna, Modeling Communications in the FIP (factory instrumentation protocol) with the Stochastic Timed Petri Model, *Proc. Of ETFA*, 1992, pp336-341.

100. S. Christensen, L.O. Jepson, Modeling and Simulation of a Network Management System Using Hierarchical Colored Petri Nets, *Proc. Of 1991 Europe Simulation Multi-Conference*, Copenhagen, Society of Computer Simulation 1991, pp47-52.

101. I. Akyildiz, *et al.*, *Stochastic Petri Net Modeling of the FDDI Network Protocol, in Protocol Specification, Testing and Verification*, *XI*, Elsevier Science Publishers B.V 1991 IFIP.

102. H. Clausen, P.R. Jensen, Validation and Performance Analysis of Network Algorithms by Colored Petri Nets, In Petri Nets and Performance Models, *Proc. Of the 5th International Workshop*, Toulouse, France 1993, pp280-289.

103. K. Jensen*, Coloured Petri Nets, Basic Concepts, Analysis Methods and Practical Use*, three volumes, Springer-Verlag 1992, 1994, and 1997.

104. G. Ciardo, *et al.,* Modeling a Scalable High-speed Interconnect with Stochastic Petri Nets, *Proc. Of the 6th* International *Workshop on Petri Nets and Performance Models*, Durham, North Carolina, October 1995.

105. R. Sahner, K. Trivedi, A. Puliafito, *Performance and Reliability Analysis of Computer Systems*, Kluwer 1996.

106. G. Chiola, A Software Package for the Analysis of Generalized Petri Nets, *Proc. Of International Workshop on Timed Petri Nets,* Torino, July 1985.

107. A. Bobbio, Petri Nets Generating Markov Reward Models for Performance/ Reliability Analysis of Degradable Systems*, Modeling Techniques and Tools for Computer Performance Evaluation*, Plenum Press 1989, pp353-365.

108. J. Couvillion, *et al.,* Performance Modeling with Ultra SAN*, IEEE Trans. Software*, V.8, 1991, pp69-80.

109. G. Ciardo, J. Muppala, K. Trivedi, SPNP Stochastic Petri Nets Package, *Proc. International Workshop on Petri Nets & Performance Model*, Kyoto, 1989, 142-150.

110. G. Rozenberg, P.S. Thiagarajan, Petri nets: Basic Notions, Structure, Behaviour, in Current Trends in Concurrency, *Lecture Notes in Computer Science 224*, Springer-Verlag, Berlin, 1986, pp.585-668.

111. P.S. Thiagarajan, Elementary Net Systems, *Petri Nets: Central Models and Their Properties, Lecture Notes in Computer Science 254*, Springer-Verlag, Berlin, 1987, pp26-59.

112. G. Rozenberg, Behaviour of Elementary Net Systems, *Petri Nets: Central Models and Their Properties, Lecture Notes in Computer Science 254*, Springer-Verlag, Berlin, 1987, pp60-94.

113. J.L. Peterson, *Petri Net Theory and the Modeling of Systems*, Prentice-Hall, Englewood Cliffs, 1981.

114. W. Reisig, *Petri Nets, EATCS Monographs on Theoretical Computer Science*, Vol.4, Springer-Verlag, Berlin, 1982.

115. H.J. Genrich, Predicate/Transition Nets, *Petri Nets: Central Models and Their Properties, Lecture Notes in Computer Science 254*, Springer-Verlag, Berlin, 1987, pp207-247.

116. F. Lee and M. Marathe, *Beyond Redundancy - A Guide to Designing High-Availability Networks*, Cisco EDCS # ENG-36854, 1999.

117. *System Hardware Availability and Reliability Calculation Worksheet*, Cisco Internal Document #702073-0000, RevA0.

118. *VoIP Availability and Reliability Model for the PacketCable Architecture*, Cable Television Laboratories Inc., PKT-TR-VoIPAR-V01-001128, 2000.

119. W. Hou, G. Okogbaa, A Simplified Availability Modeling Tool for Networks with 1:1 Redundant Software-Hardware Systems, *Proceedings of Annual Reliability and Maintainability Symposium (RAMS)*, 2002, pp569 - 576.

120. W. Hou, *High Availability Analysis for Cactus Solution (1.0R)*, Cisco EDCS #ENG-105749, 2001.

121. W. Hou, *Cactus 1.0R End to End Availability Model*, Cisco EDCS #ENG-108451, 2001.

# APPENDICES

Appendix 1 SAMOT Modules

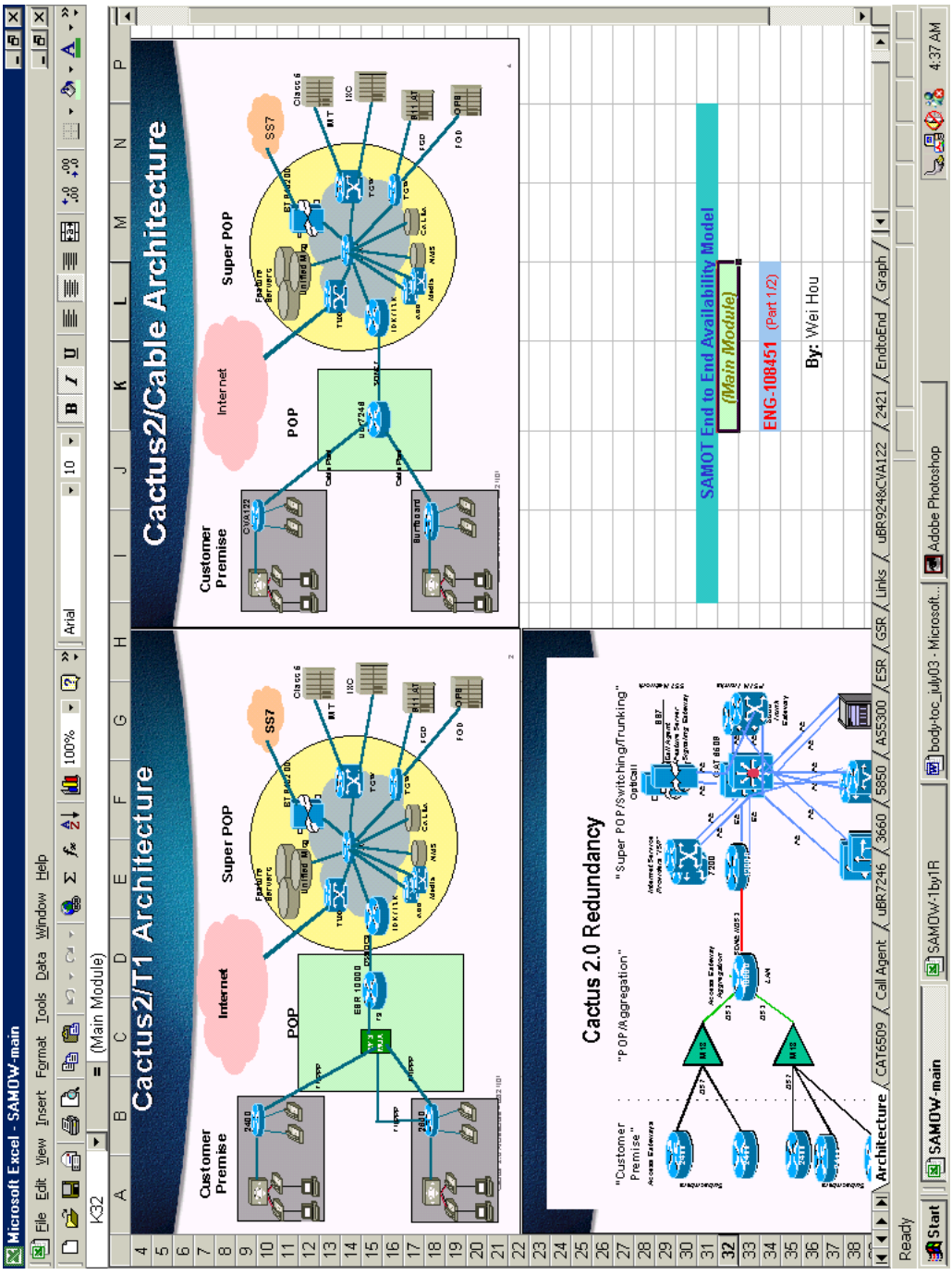



Figure A-1.1. SAMOT-Main Module: Solution Architectural Scenarios

Appendix 1 (Continued)

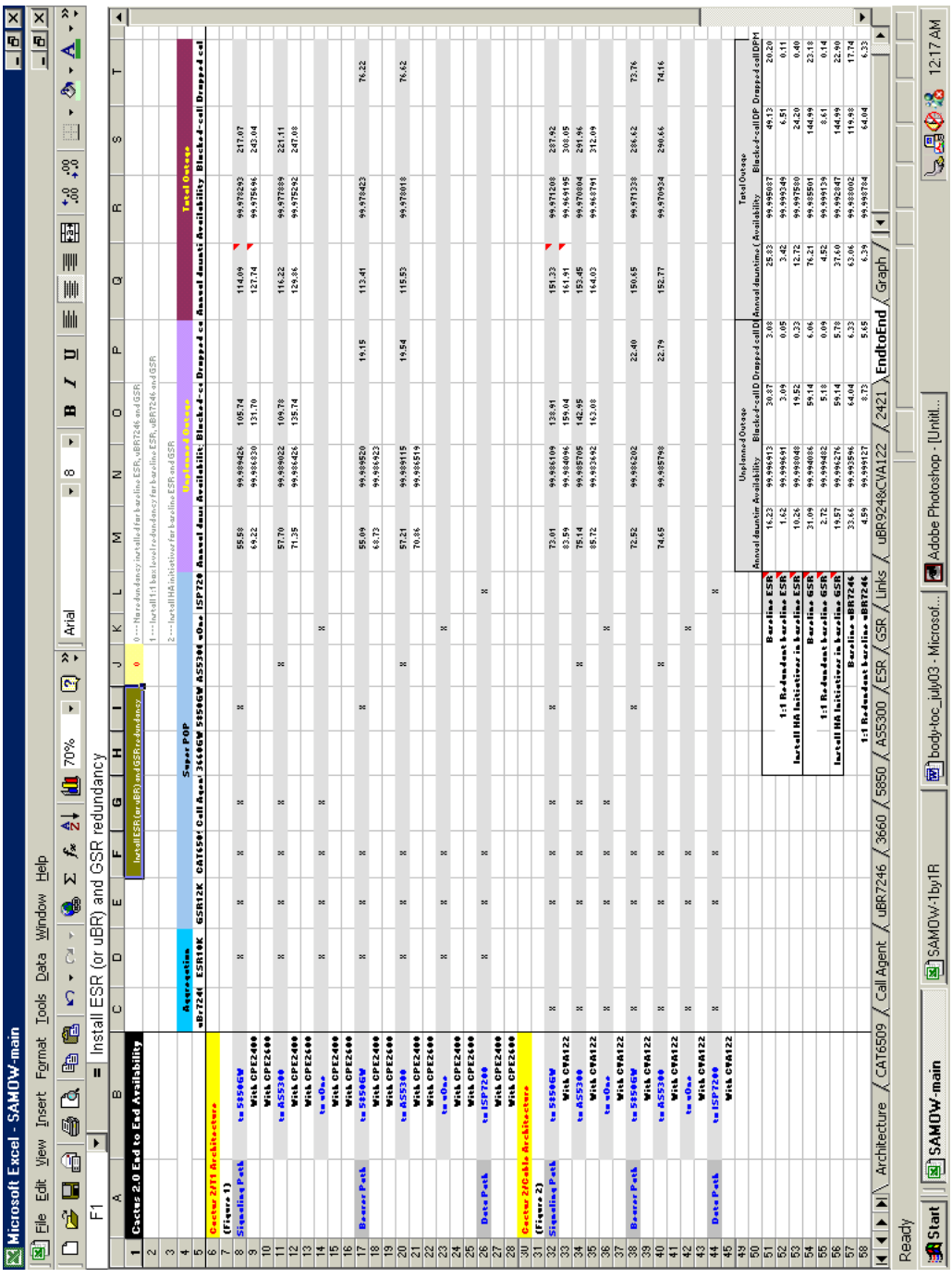


Figure A-1.2. SMOT-Main Module: End-to-End Availability Worksheet

Appendix 1. (Continued)

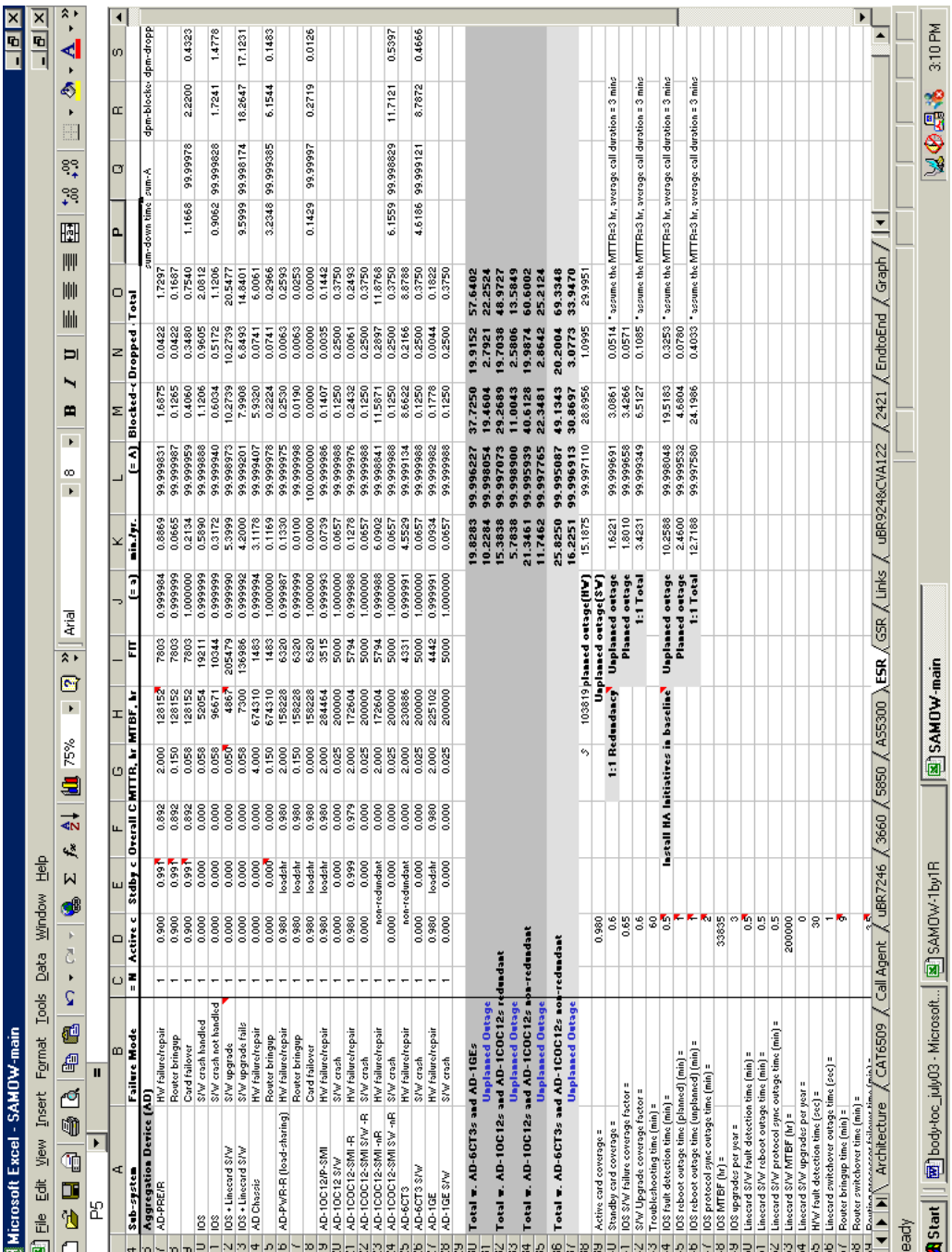


Figure A-1.3. SAMOT-Main Module: Aggregation Device

Appendix 1. (Continued)

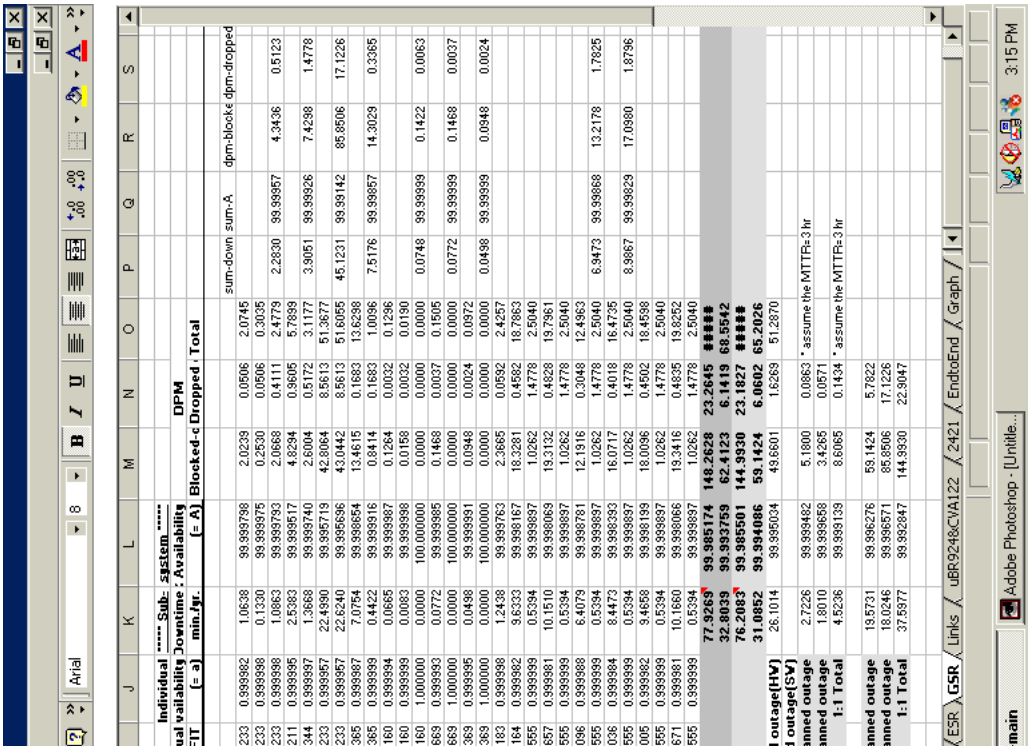



Figure A-1.4. SAMOT-Main Module: Core Router

Appendix 1. (Continued)

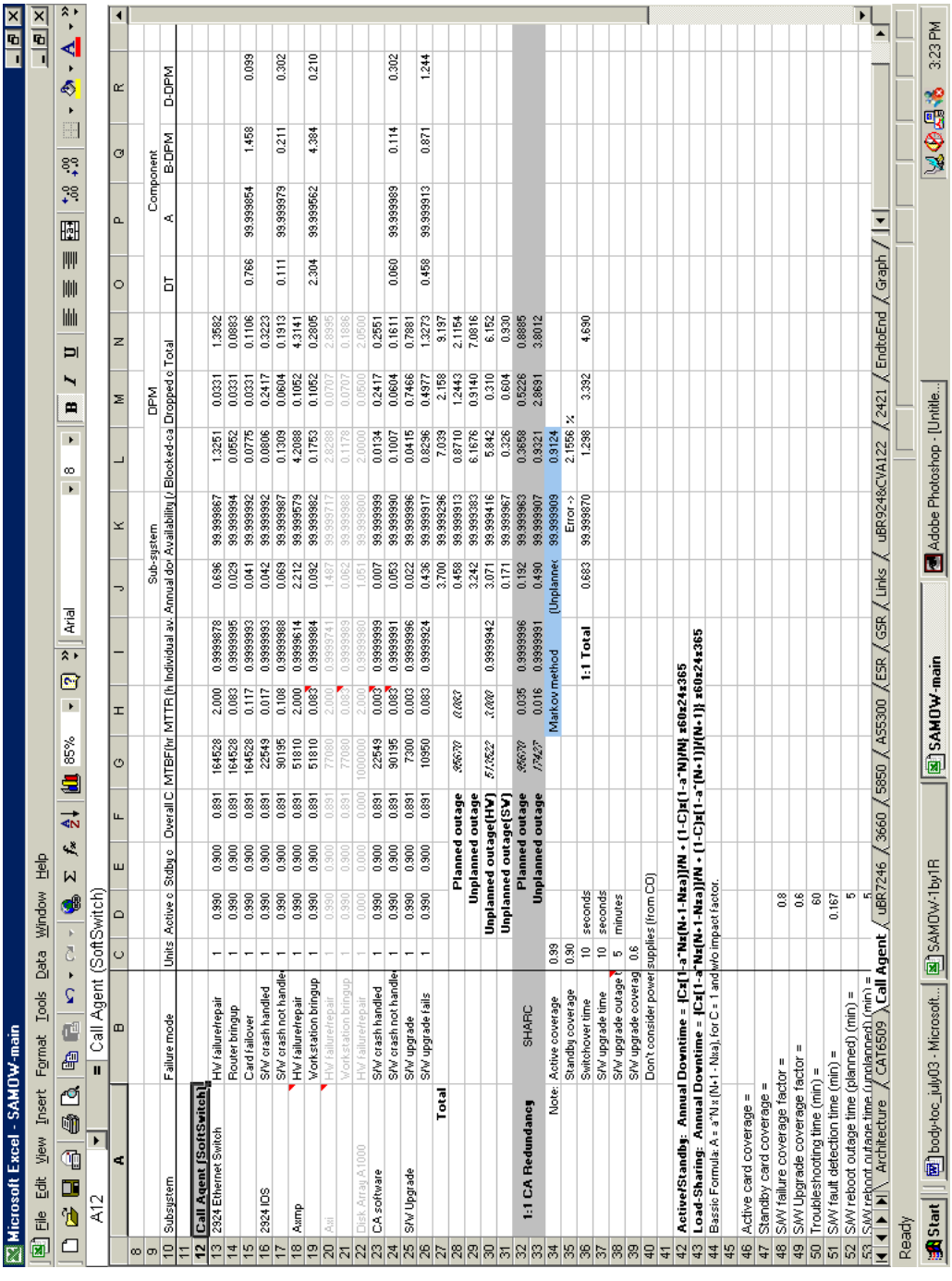



Figure A-1.5. SAMOT-Main Module: Softswitch System

Appendix 1. (Continued)

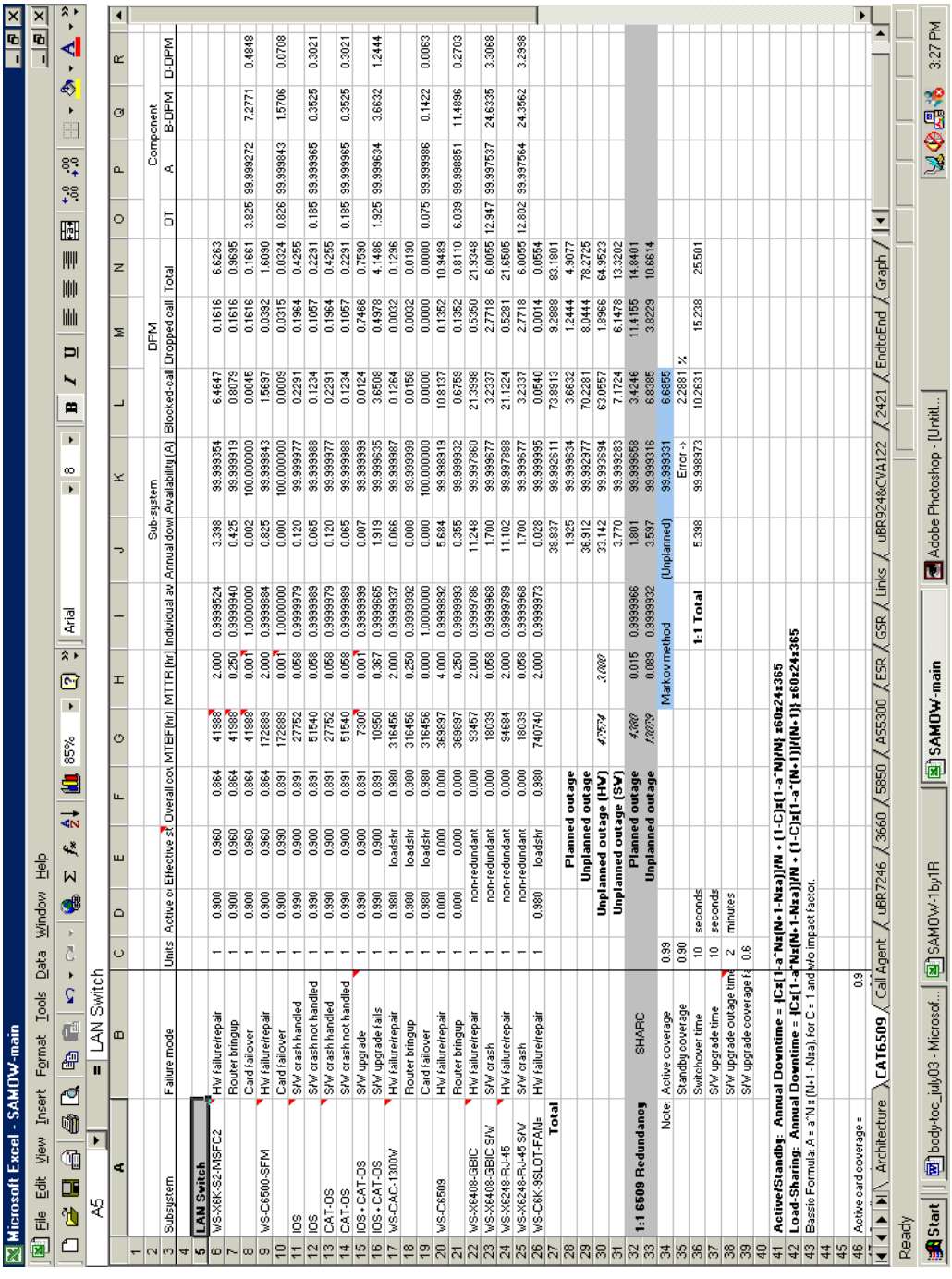



Figure A-1.6. SAMOT-Main Module: LAN Switch

Appendix 1. (Continued)

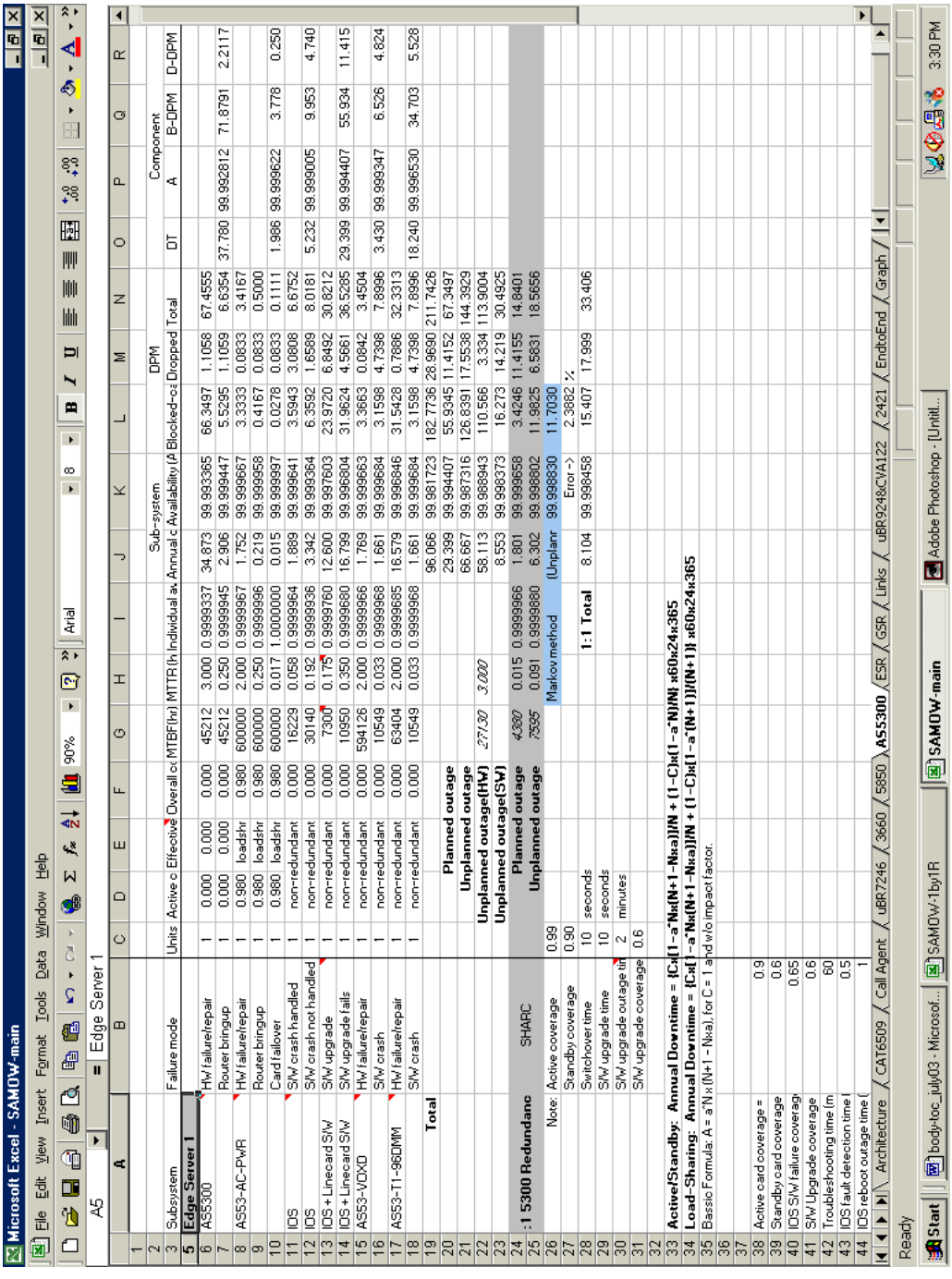
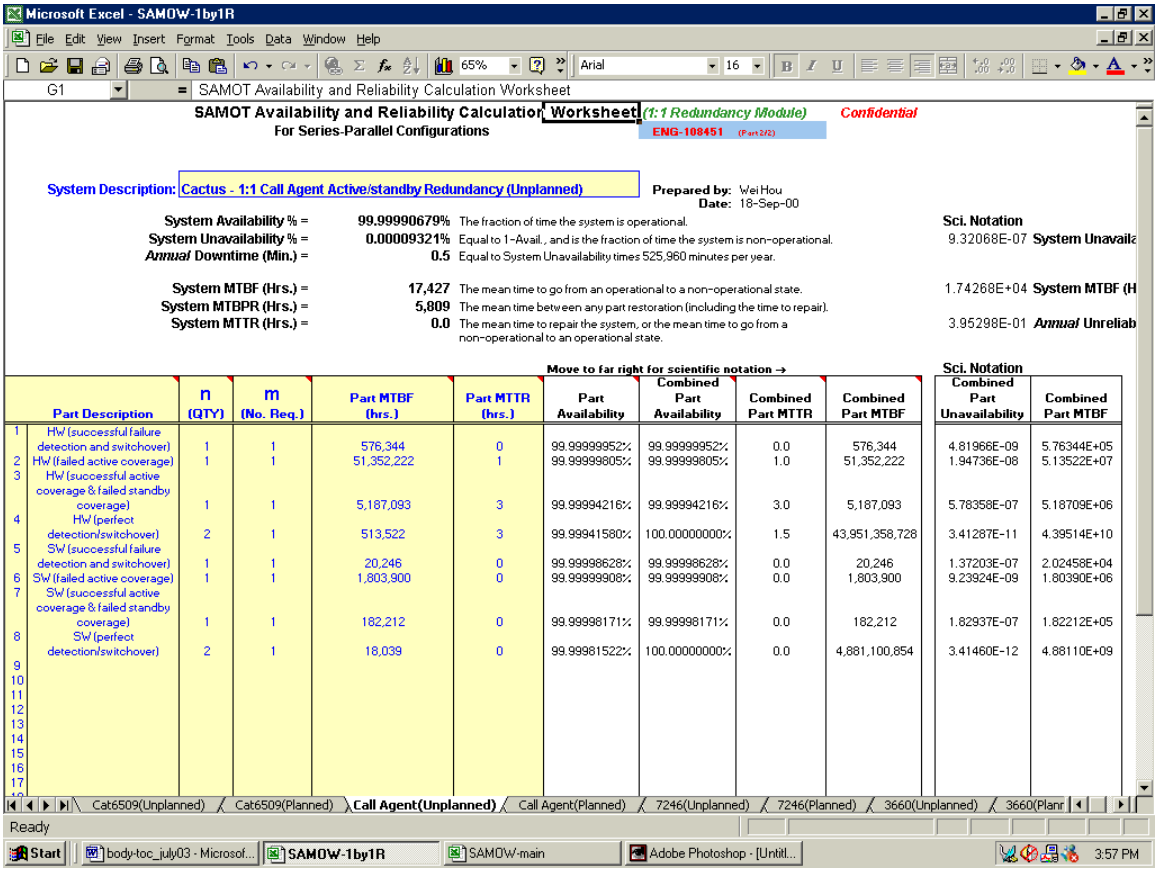



Figure A-1.7. SAMOT-Main Module: Edge Server 1

Appendix 1. (Continued)



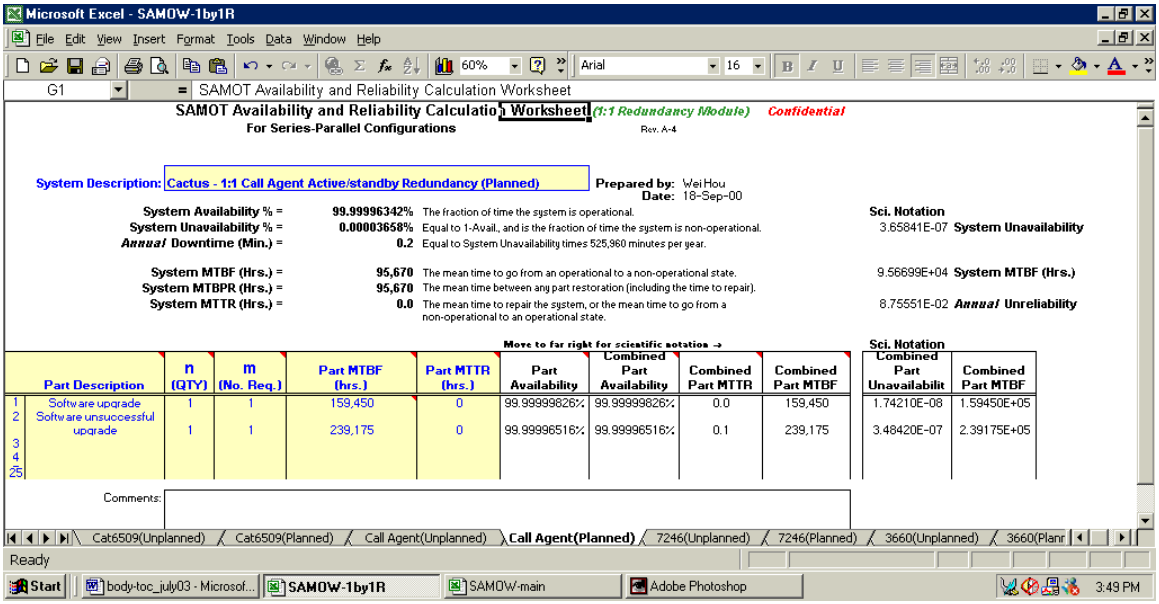




Figure A-1.8. SAMOT-1:1 Redundancy Module: SoftSwitch

Appendix 1. (Continued)

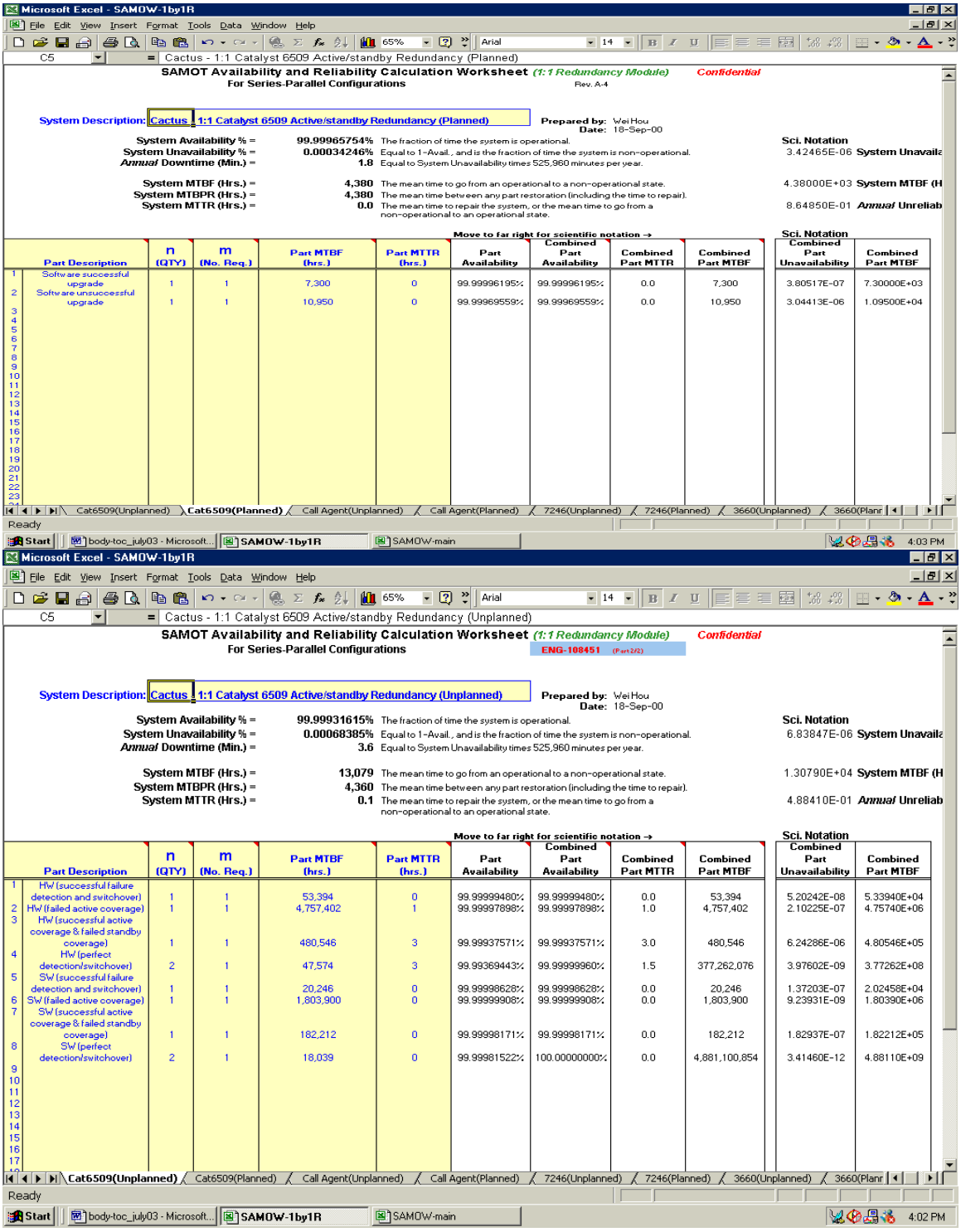



Figure A-1.9. SAMOT-1:1 Redundancy Module: LAN Switch

Appendix 1. (Continued)

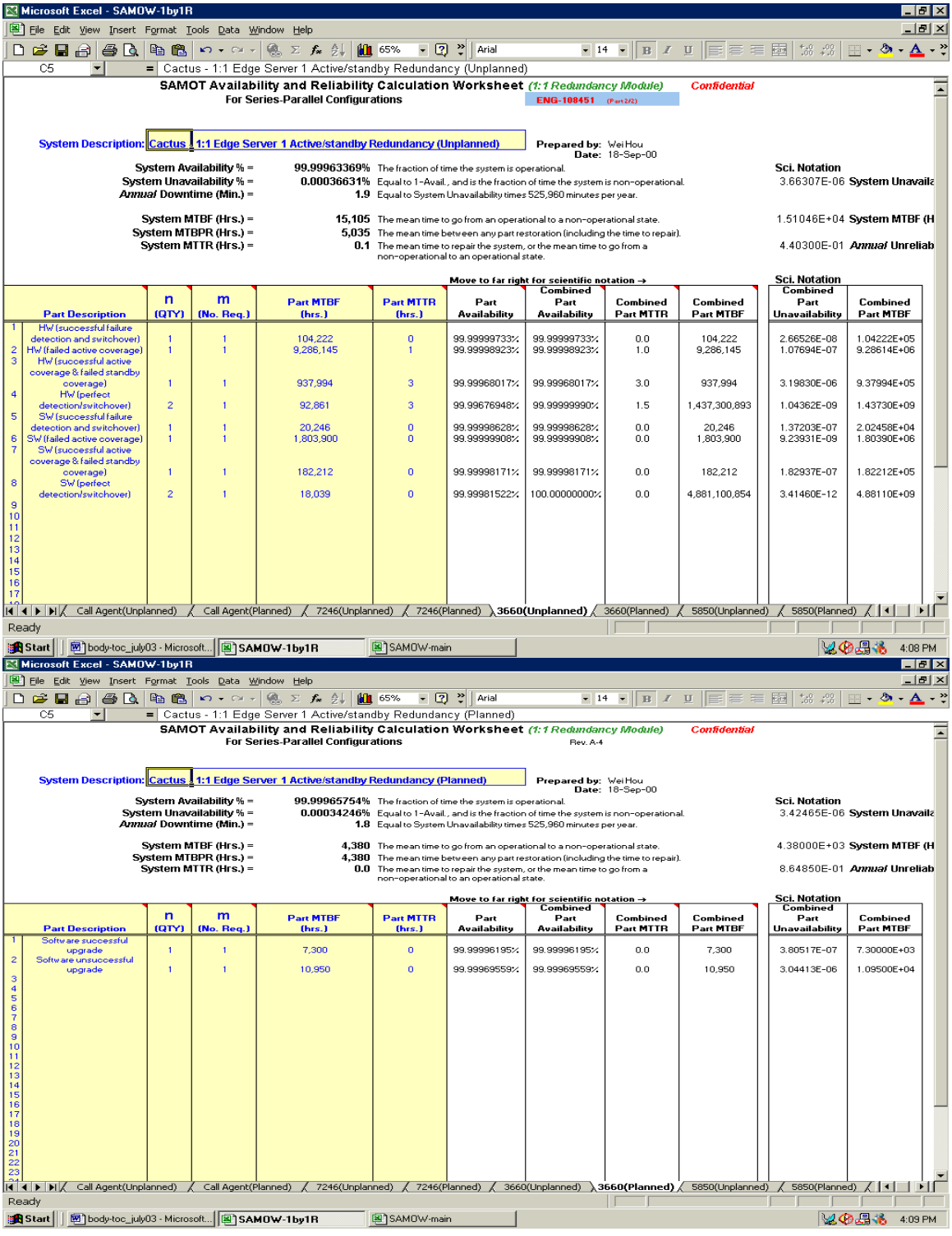



Figure A-1.10. SAMOT-1:1 Redundancy Module: Edge Server 1

Appendix 2 Markov Analysis Tool

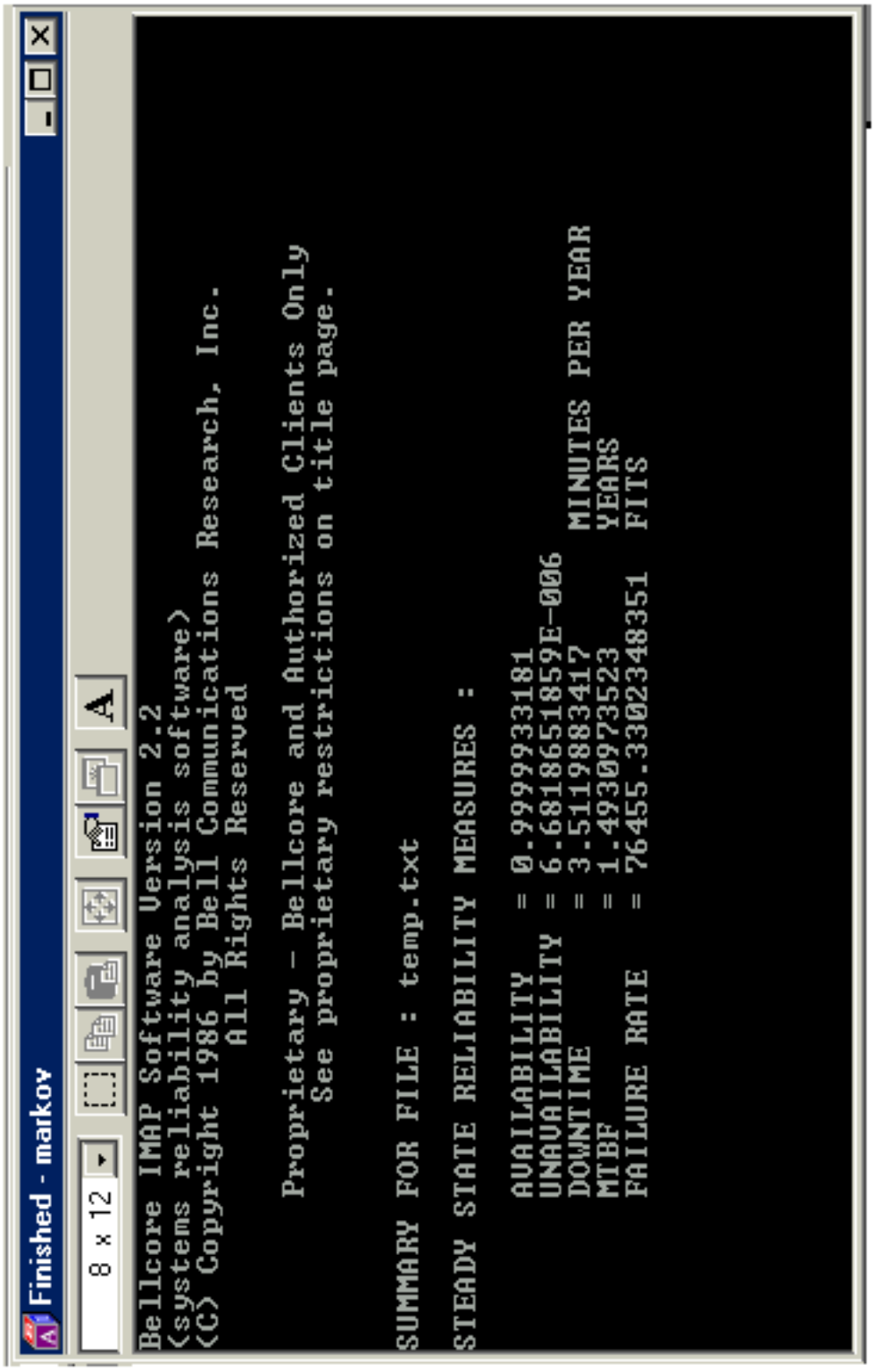



Figure A-2.1. Markov Analysis Summary Demo

## Appendix 2.1. Markov Analysis Input File

```
Input File Name: sample1.txt
=====================
# 1:1 Active/Standby Hardware + Software Redundancy
# Variables: FIT rates, MTTR, coverage factors, switch time

states = 13
failed = 2,4,5,6,10,11

# Parameters:

MTTFH = 47574                 # HW Mean Time To Failure (hr)
MTTFS = 18039                 # SW Mean Time To Failure (hr)
lambdaH = 1/MTTFH             # HW Failure rate of active unit
lambdaS = 1/MTTFS             # SW Failure rate of standby unit
SwitchTimeH = 10              # HW Switchover time to standby (sec)
SwitchTimeS = 10              # SW Switchover time to standby (sec)
betaH = 1/(SwitchTimeH/3600)  # HW Switchover rate
betaS = 1/(SwitchTimeS/3600)  # SW Switchover rate
MTTR1H = 10/60/60             # MTTR of HW unit non-service failures (hr)
MTTR1S = 10/60/60             # MTTR of SW unit non-service failures (hr)
MTTR2H = 3                    # MTTR of HW unit service failures (hr)
MTTR2S = 2/60                 # MTTR of SW unit service failures (hr)
mu1H = 1/MTTR1H               # Mean HW repair rate for non-service affecting failures
mu1S = 1/MTTR1S               # Mean SW repair rate for non-service affecting failures
mu2H = 1/MTTR2H               # Mean HW repair rate for service affecting failures
mu2S = 1/MTTR2S               # Mean SW repair rate for service affecting failures
c1 = 0.99                     # Coverage factor of active unit
c2 = 0.90                     # Coverage factor of standby unit

# Transitions:

## States for detected failures
1       2       c1*lambdaH
2       3       betaH
3       1       mu1H
3       4       lambdaH
4       1       mu2H
3       5       lambdaS
5       1       mu2S
1       6       c1*lambdaS
6       7       betaS
7       1       mu1S
7       4       lambdaH
7       5       lambdaS
1       8       c2*lambdaH
8       1       mu1H
8       4       lambdaH
```

Appendix 2. (Continued)

```
8       5       lambdaS
1       9       c2*lambdaS
9       1       mu1S
9       4       lambdaH
9       5       lambdaS

## States for undetected failures
1       10      (1-c1)*lambdaH
10      1       mu2H
1       11      (1-c1)*lambdaS
11      1       mu2S
1       12      (1-c2)*lambdaH
12      4       lambdaH
12      5       lambdaS
1       13      (1-c2)*lambdaS
13      4       lambdaH
13      5       lambdaS
```

Appendix 2. (Continued)

Appendix 2.2. Markov Analysis Output File

MARKOV MODEL SOLUTION FOR STEADY STATE AVAILABILITY, (V 2.2) JULY 1986
BELL COMMUNICATIONS RESEARCH, INC.

MODEL PARAMETERS :

| | |
|---|---|
| MTTFH | = 47574 |
| MTTFS | = 18039 |
| lambdaH | = 2.101988E-005 |
| lambdaS | = 5.543545E-005 |
| SwitchTimeH | = 10 |
| SwitchTimeS | = 10 |
| betaH | = 360 |
| betaS | = 360 |
| MTTR1H | = 0.002778 |
| MTTR1S | = 0.002778 |
| MTTR2H | = 3 |
| MTTR2S | = 0.033333 |
| mu1H | = 360 |
| mu1S | = 360 |
| mu2H | = 0.333333 |
| mu2S | = 30 |
| c1 | = 0.99 |
| c2 | = 0.9 |

STATE PROBABILITIES :

| STATE | PROBABILITY | MINUTES/YR | |
|---|---|---|---|
| 1 | 0.909084503 | 4.77815E+005 | |
| 2 | 5.254934172E-008 | 0.02762 | * FAILED STATE |
| 3 | 5.254933056E-008 | 0.02762 | |
| 4 | 5.732678471E-006 | 3.0131 | * FAILED STATE |
| 5 | 1.679856888E-007 | 0.08829 | * FAILED STATE |
| 6 | 1.385876370E-007 | 0.07284 | * FAILED STATE |
| 7 | 1.385876075E-007 | 0.07284 | |
| 8 | 4.777211869E-008 | 0.02511 | |
| 9 | 1.259887341E-007 | 0.06622 | |
| 10 | 5.732655461E-007 | 0.30131 | * FAILED STATE |
| 11 | 1.679850145E-008 | 0.00883 | * FAILED STATE |
| 12 | 0.024993485 | 13136.57574 | |
| 13 | 0.065914965 | 34644.90573 | |

STEADY STATE RELIABILITY MEASURES:

| | | |
|---|---|---|
| AVAILABILITY | = 0.9999933181 | |
| UNAVAILABILITY | = 6.6818651859E-006 | |
| DOWNTIME | = 3.5119883417 | MINUTES PER YEAR |
| MTBF | = 1.4930973523 | YEARS |
| FAILURE RATE | = 76455.3302348351 | FITS |

Appendix 3 MORIN Algorithm

Here are codes implementing the MORIN reliability calculation.

```
/* ********************************************************************** *
 *                                                                        *
 *  MORIN_RCal.c                                                          *
 *                                                                        *
 *  This program is to to calculate the network reliability based         *
 *  on the reliability of each node and link along the event trees.       *
 *  This program is designed to run on sunblast.eng.usf.edu               *
 *                                                                        *
 *   Code designed and created by W. Hou                                  *
 * ********************************************************************** */

#include <stdio.h>
#include <string.h>
#include <iostream.h>
#include <sys/types.h>
#include <netdb.h>

#define node_number 4
#define link_number 5


main()          /* calculate network reliability based on generated event trees */


{
        char  event_tree[];       /* the event tree path sets      */
        char  node, link;         /* the node index, link index    */
        char  link;               /* the link index                */
        double R_node[];          /* the node reliability          */
        double RM_node[];         /* the node's modified reliability   */
        double RMo_node[];        /* the node's modified reliability with operational incoming links */
        double RMf_node[];        /* the node's modified reliability with failed incoming links  */
        double R_link[];          /* the link reliability          */
        double R_source[];        /* source node reliability       */
        double R_event_tree[];    /* event tree reliability        */
        double R                  /* overall network reliability   */
        double Rh_node[];         /* the node hardware reliability         */
        double Rh_link[];         /* the link hardware reliability         */
        double Rh_source[];       /* source node hardware reliability      */
        double Rs_node[];         /* the node software reliability         */
        double Rs_source[];       /* source node software reliability      */
        double R = 1;             /* the initial network reliability       */
        double RMo_node = 1;      /* the initial modified node reliability with
                                     operational incoming links     */
        double RMf_node = 1;      /* the initial modified node reliability with failed incoming links */
        int i, j, k;              /* node j and link k on event tree i     */
```

Appendix 3. (Continued)

```
        while ((event_tree = getchar()) != EOF)

                for (i = 0; i < event_tree number; ++i )
                 {

                        for (j = 0; j < node number on the event tree; ++j )

                        {

                            for (k = 0; k < adjacent links to node j; ++k)

                                if (link[k]_adjacent = OPERATIONAL)
                                   RMo_node [j] = R_node[i] * R_link[k];

                                else
                                   RMf_node [j] = (1-R_node [j]) + R_node[j] * (1 - R_link[k]) ;

                         R_event_tree[i] = R_source * RMo_node [j] * RMf_node [j];
                        }

                Printf("reliability of event tree i is :", R_event_tree[i]);
                }

                R *= R_event_tree[i];
                }
        Printf("overall reliability is :", R)

}

/* codes for ET generating and other modules are available upon NDA  */
```

## ABOUT THE AUTHOR

Wei Hou received a BS degree (1989) in Telecommunication Management Engineering and a MS (1992) in Systems Engineering both from Beijing University of Posts and Telecommunications, China. After his Master's graduation, Mr. Hou served the Ministry of Information Industry of China as a research staff before he joined Ericsson as a system engineer. He led a six-month consulting project at Quality Assurance of GE Medical Systems Information Technology in 1996 shortly after his arrival at USA and co-oped as an IT engineer at the Availability Management Center of Verizon in 1997 and 1998. Mr. Hou has worked as an availability analyst at Systems and Solutions Engineering of Cisco Systems from 2000 to 2001 and since then he has been with Sun Microsystems as a hardware member of technical staff.

During his doctorate research co-sponsored by National Science Foundation (Award # DMII 9500289) and Department of Industrial and Management Systems Engineering at USF, Wei Hou has presented and published a number of papers and tutorials in international symposiums and conferences. He has also authored over a dozen of technical reports for GE, Verizon, Cisco, and Sun Microsystems. He was a student member of IIE and INFORMS, a member of IEEE Reliability Society, a member of International WHO'S WHO.