Invited paper, accepted for publication in
Reliability Engineering and System Safety

Stochastic Petri Nets for the Reliability Analysis of Communication Network Applications with Alternate-Routing

Meera Balakrishnan¹ and Kishor S. Trivedi²

Department of Electrical Engineering

Duke University, Box 90291

Durham, NC 27708-0291.

March 3, 1995

¹Supported by an NSF Postdoctoral Research Associate Grant No. 92-120.

²Supported in part by NSF Grant No. CCR-9108114.

Abstract

In this paper, we present a comparative reliability analysis of an application on a corporate B-ISDN network under various alternate-routing protocols. For simple cases, the reliability problem can be cast into fault-tree models and solved rapidly by means of known methods. For more complex scenarios, state space (Markov) models are required. However, generation of large state space models can get very labor intensive and error prone. We advocate the use of *stochastic reward nets* (a variant of stochastic Petri nets) for the concise specification, automated generation and solution of alternate-routing protocols in networks. This paper is written in a tutorial style so as to make it accessible to a large audience.

1 Introduction

Providers of telecommunication services generally offer network services to corporate customers for interconnecting office locations over their *own* network. Such users lease bandwidth and are charged a fixed dollar amount independent of usage, making it economically attractive to corporate customers. Furthermore, individual corporate networks can be engineered independently of other networks sharing the network facilities: in particular, they can be made more reliable. A corporate network on the future Broadband Integrated Services Network (B-ISDN), a *virtual network*, is likely to be the choice for interconnecting various plants of a power company, for example. In this paper, we present a comparative reliability analysis of an application on such a network under various alternate-routing protocols.

When viewed at the *connection level*, the B-ISDN is a dynamic system with built-in detection and reconfiguration mechanisms. Traffic conditions and hardware status are constantly monitored and reported to a centralized or distributed network management subsystem. Fault-tolerance is incorporated through a dynamic re-routing of traffic over alternate routes if the primary route fails. Failures may be caused by an optical-fiber cut or by network congestion culminating in unacceptable performance for a given connection.

In this paper, we model and analyze for the reliability of an application on a corporate network under three alternate-routing protocols. The goal is to predict the effect of candidate routing policies on the reliability of the network application. Thus, for each policy, we determine an appropriate model such as a fault-tree model or a homogeneous continuous-time Markov chain model and use corresponding solution techniques to compute reliability. The fault-tree is an example of a model type solvable by *implicit* state space methods while the Markov chain is an example of a model type that uses *explicit* state space generation and solution methods. Reliability block diagram and traditional network reliability models can be categorized as using implicit state space methods. We shall discuss the capabilities and limitations of these models through examples in Section 2. We consider cases of non-repairable as well as repairable links.

A very powerful construct for specifying systems with concurrent and asynchronous activities is the Petri net [34], i.e., it has the capability of describing complex discrete-event systems. In the context of our problem, failure and repair *activities* of each link are concurrent, while alternate-routing protocols are inherently event driven (asynchronous). While a Petri net essentially describes a state-machine with state transitions triggered by discrete events, it does not specify the time spent

in any state. Failure and repair *characteristics* on the other hand, are governed by the elapsed time between events: for example, in a network with link-repair facilities, the time-to-failure of a link is the interval between the event that *link is repaired* and the event that *link fails*. Therefore, by associating distribution functions to appropriate transitions, it is possible to specify failure and repair characteristics in a Petri net. Such transitions are referred to as *timed transitions* while non-timed transitions are called *immediate transitions*. When all timed transitions have associated exponentially distributed characteristics, the Petri net is referred to as a Generalized Stochastic Petri net (GSPN) [30]; note that it is possible for a GSPN to have no immediate transitions. The stochastic process underlying a GSPN is a homogeneous continuous-time Markov chain and solution methods for Markov chains apply. Indeed, a fault-tree model which assumes exponentially distributed characteristics may also be specified as a GSPN and solved by an explicit state space method. The stochastic process underlying such a fault-tree is a Markov chain with a special *form*. So for these special cases, state space generation and a corresponding solution can be avoided altogether and the problem of reliability prediction becomes one of visiting the up-states and accumulating the associated state probabilities. That is, neither a Markov chain nor a stochastic Petri net specification is really required. When the time-to-failure and/or time-for-repair are not exponentially distributed, the underlying process may be a time-non-homogeneous Markov chain [19], a semi-Markov process [13], a Markov regenerative process [26], or a generalized semi-Markov process [21], and specialized solution methods can be used. Fault-tree-like methods can sometimes accommodate non-exponential time-to-failure and time-for-repair distributions. More generally, however, non-Markovian Petri nets can be used [9, 21]. Some non-Markovian Petri nets can be solved analytically; otherwise, the Petri net is solved by discrete event simulation.

In practice, a Petri net representation is attractive because it automates the solution process. For example, the underlying Markov chain of a GSPN can be automatically generated from its Petri net specification and solved for the measure of interest without user intervention [14]. In fact, there is a systematic method of describing the measure of interest through specification of *reward rates* in a stochastic Petri net; reliability, availability and performability measures are easily obtained by suitable reward assignments. The Stochastic Petri net is then referred to as a Stochastic Reward net (SRN). The automatic generation capability is particularly advantageous when the state space of the Markov chain model is too large to manually input to a Markov chain solver. One option is to generate the Markov chain by a program, but this requires the user to translate the problem into a Markov chain description. The Petri net allows the problem to be specified at a higher level and automatically generates the Markov chain, and in addition, this specification is independent

of the solution method. Several tools supporting the automatic analysis of GSPNs and their variants are available. These include GreatSPN [8], METASAN [39], UltraSAN [17], SPNP [10] and DSPNexpress [28]. For an exposition on other specification methods see [22].

The models we consider in this paper are Markov models and correspondingly we use the Stochastic Petri Net Package (SPNP) which accepts an SRN input. In general, a graphical Petri net representation for a real system gets too complicated and cumbersome for clear pictorial comprehension. SPNP therefore offers a high-level Petri net language, the *C-based Stochastic Petri net Language* (CSPL), for ease of specification [11]. The advantage is twofold. First, it allows the user to specify sub-Petri nets, and to express the interaction between the subnets without the explicit drawing of arcs. Secondly, complicated logical functions for the enabling of transitions can be specified more naturally via boolean expressions in CSPL rather than by (sometimes) awkward and unwieldy pictorial rendering of arcs. The advantages offered by SPNP can be exploited to the fullest by a judicious decomposition which preserves the visual advantage offered by Petri net arcs while simultaneously removing clutter. Figure 10 shows the subnets we have used to specify some of the alternate-routing protocols presented in this paper.

The paper is organized as follows. In Section 2, we present some fault-tree models of reliability and give an example where the fault-tree model is inadequate. In Section 3 we introduce stochastic Petri nets and reward nets. In Section 4, we describe the architecture of a corporate network and in Section 5 we model alternate-routing protocols to obtain reliability for an application. Conclusions are in Section 6.

2 Some Issues in Reliability Modeling

In this section we consider three simplified models of alternate-routing in a network to demonstrate several issues in reliability modeling. The first two models do not permit repair; in the third, link-repair is allowed. The network is represented by a graph whose nodes denote the office locations of a corporation. Edges of the graph represent communication links between office locations. The measure of interest is *reliability*, $R(t)$, a measure of the network's ability to maintain a given set of connections continuously over a given interval $(0, t]$. We will assume that the network is *up* at $t = 0$; the network is assumed to be *up* if the given set of node-pairs is connected either by a direct link or by a two-link alternate path (see Figure 1). Formally, reliability is the probability that the network is continuously *up* over an interval $(0, t]$, assuming a suitable initial condition. In this

paper, we restrict the time-to-failure and time-for-repair to be exponentially distributed. Through examples in this section, we seek to present (i) situations where implicit state-space methods can be used. (Unfortunately, there are currently no automated tests for detecting these situations.) (ii) situations where explicit state-space methods are needed. Examples 1 and 2 below can be solved by implicit state-space methods. Example 3 requires explicit state-space methods.

Example 1: In Figure 1, the network is *up* whenever node-pairs A-B and C-D are both connected, either directly, or by the two-link alternate routes listed. We impose the condition that the alternate routes of the node-pair A-B should be disjoint from those of node-pair C-D. Failure processes are mutually independent. Our goal is to obtain reliability as per the definition above.

To list the essential features of a fault-tree analysis, the criterion for partitioning the state space into the *down*-states and the *up*-states is given by the fault-tree (Figure 1). Since failed links are not repairable, the status of the links at time t is sufficient information to infer that the network has been operating continuously throughout $(0, t]$. Thus, the problem of unreliability/reliability evaluation by this implicit state-space method is that of visiting each *down/up*-state exactly once and accumulating the associated probability of state occurrence.

Enumeration is impractical: the reliability problem for implicit state-space methods is shown to be #P-complete [16]. A vast body of literature addresses the problem of managing the process of accumulation by dividing the state space into subsets and accumulating the probability associated with states in each subset [5, 27, 41, 35]. Often the subsets are not mutually disjoint and fault-tree methods and other implicit state-space reliability analysis methods make adjustments for visiting a state more than once. The standard procedure for obtaining $P\{\overline{Netup}\}$ for the current example would be to systematically process the AND and OR gates. A sufficient condition for this procedure to be applicable is for the fault-tree to have AND, OR and/or K-of-N gates and for basic events to be statistically independent and distinct. For the example in Figure 1 the network unreliability is give by:

$$\begin{aligned}
\overline{R}_{netup} &= 1 - R_{AB} R_{CD} \\
\overline{R}_{AB} &= \overline{R}_{ab} \overline{R}_{AB1} \overline{R}_{AB2} \overline{R}_{AB3} \\
\overline{R}_{CD} &= \overline{R}_{cd} \overline{R}_{CD1} \\
\overline{R}_{AB1} &= 1 - R_{ac} R_{cb}, \\
&\text{and similarly for other alternate paths.}
\end{aligned}$$

Although gate-processing is a very efficient solution method, it has restricted applicability. A more general approach is to obtain $P\{\overline{Netup}\}$ from the expression for $\overline{Netup}$.

Example 2: This example permits common-mode failures, a case where the gate-processing algorithm fails. Shared link $B-C$ in Figure 2 is responsible for the common mode failure and appears as a *repeated event* in the fault tree.

Algorithms for fault-tree analysis with repeated events first obtain minimum cut sets or *mincuts* [16], then process these to obtain mutually disjoint events. A mincut is a term (set) in the minimized-sum-of-products expression of the fault-tree's *structure function*. The mincuts are not guaranteed to be disjoint and so are processed to yield mutually disjoint sets, causing the associated events in the underlying probability space to be mutually disjoint. A *sum-of-disjoint-products* form can be obtained by Boolean manipulation of the minimized-sum-of-products form of $\overline{Netup}$ [16, 36, 38]. For Example 2, network unreliability is given by:

$$\begin{aligned} P\{\overline{Netup}\} = & (1 - R_{ab}) \cdot (1 - R_{ac}) + (1 - R_{bc}) \cdot (1 - R_{ab}) \cdot R_{ac} + (1 - R_{bc}) \cdot (1 - R_{cd}) \cdot R_{ab} \\ & + R_{bc} \cdot (1 - R_{ab}) \cdot (1 - R_{bd}) \cdot (1 - R_{cd}) \cdot R_{ac} + R_{bc} \cdot (1 - R_{bd}) \cdot (1 - R_{cd}) \cdot R_{ab} \end{aligned}$$

Note that probabilities in the above expression vary with time; but, since the fault-tree analysis is valid for every t -value, the time dependence is often suppressed in implicit state space reliability expressions. The stochastic process underlying a fault-tree (with exponential time-to-failure distributions) is a Markov chain on the state space described earlier, and it is possible to solve for reliability using Markov chain methods after an explicit state-space enumeration. Clearly, this is not a practical approach and implicit state-space methods should be used whenever possible. For models (as in Example 3, below) that cannot be solved by implicit state-space methods, possible approaches for handling large state-spaces are state-aggregation or lumping [25, 2, 33, 3], hierarchical decomposition [24] and state truncation [32].

Example 3: For this example, we return to the model in Example 1, but allow each link to have its own repair facility. Failure and repair processes are mutually independent.

At any time instant, a link is either *up* or *down*. The definition of the set of *down*-states and the set of *up*-states of the network does not change with time and is described by the fault-tree structure. However, reliability can no longer be obtained by a fault-tree set-up designed to yield instantaneous state-occurrence probabilities because the repair process needs to be controlled on a (global/network) state-dependent basis for obtaining the reliability measure; the difficulty is that

the information that the network is *up* at time instant t does not guarantee that the network was *up* continuously over the interval $(0, t]$, for the network could have failed and come up one or more times in this interval. Note that a two-state Markov chain model for each link used in conjunction with a fault-tree will yield *instantaneous (un)availability*, but not network (un)reliability because the time-varying solution of Markov chains at the link level give the instantaneous link-(un)availability. In an explicit state space model, state-dependent control on the repair process is easily achieved by modifying the original stochastic process to forbid the repair of links when the network fails for the first time after time zero. Thus, Markov reliability models have an absorbing subset of states consisting of all the down-states of the system. Sample paths of the original process and the modified version are shown in Figure 3 and Figure 4, respectively. The absorbing Markov chain of the modified process is solved for instantaneous (time-varying) state probabilities; network reliability is obtained by summing the state probabilities of the up-states.

3 Stochastic Petri Nets and Reward Nets

In this section, we introduce the stochastic Petri net through Example 2 of the previous section. As discussed therein, this example is solvable by an implicit state space method. However, here we use the more elaborate explicit state space approach for introducing the stochastic reward net. For an introduction to the original Petri net, we refer the reader to [34]. As mentioned in Section 1, underlying a stochastic reward net is a Markov chain. By associating *reward rates* to the states of the Markov chain, we can construct a Markov reward model and obtain useful measures of the system being modeled. Consequently, a Markov reward model is specified by a stochastic Petri net along with a reward specification and the combined specification is termed Stochastic Reward Net or SRN. The Stochastic Petri Net Package (SPNP) [11] allows the specification of SRN models and efficient, numerically stable algorithms employing sparse matrix techniques are used to solve the underlying Markov chain [37]. In this section we introduce stochastic Petri nets and discuss Markov reward models.

Figure 5 shows an SRN, a bipartite graph. The graph or net contains two types of nodes: *places* and *transitions*. Places represent *conditions* and are shown as circles in the figure; places $AB1up$, $CD1up$, $AB1dn$ and $CD1dn$ denote the up/down condition of alternate routes $AB1$ and $CD1$ (see Figure 2 for route descriptions). Transitions represent events that occur in the network. *Timed-transitions*, $T-AB1fl$ and $T-CD1fl$, shown in the figure by unfilled rectangles, represent the failure

(event) of alternate paths $AB1$ and $CD1$, respectively, due to failure of the unshared link. $T-bth$ represents the simultaneous failure of $AB1$ and $CD1$ due to failure of the shared link. $T-ABnwfl$ and $T-CDnwfl$ are *immediate transitions* and are represented by bars in Figure 5. Immediate transitions *fire* immediately when the transition is enabled. Timed-transitions fire after a delay decided by a sample drawn (at the enabling instant) from the distribution specified for the transition. The failure activities of the direct links are shown by the remaining places, transitions and arcs in Figure 5. Place $Netdn$ denotes the condition of network failure. *Arcs*, shown by arrows, connect places to transitions and transitions to places. An arc *from* a place *to* a transition is called an *input arc* to the transition; the corresponding place is an *input place* for the transition. Similarly, an arc *from* a transition *to* a place is called an *output arc*; the corresponding place is an output place for the transition. Arcs connecting places to places or transitions to transitions are not allowed.

As mentioned in [34], a Petri net *graph* is like a flowchart for a computer program and Petri nets are *executed* through the movement of *tokens*. One or more tokens, denoted by black dots on the graph, occupy places. Tokens *enable* transitions by their presence in associated input places. The presence/absence of tokens correspond to satisfaction/non-satisfaction of the condition represented by the place. A transition with more than one input place can fire only if enabled by tokens at every input place. For example, $T-ABnwfl$ in Figure 5 will fire only if both routes between node-pair A-B are down. Enabled transitions fire and move tokens. A transition fires by removing a certain (input arc-cardinality dependent) number of tokens from each input place connected to it and generating new (output arc-cardinality dependent number of) tokens which are deposited into each output place connected to it. In Figure 5, transitions are enabled by single tokens residing in corresponding input places; the default arc-cardinality is 1. The distribution of tokens in the Petri net is said to be its *marking* and a marking is represented by a vector of the number of tokens in each place of the Petri net. The initial marking shown in Figure 5 is a 9-tuple:

$$\begin{aligned} & \#ab - up, \#ab - dn, \#AB1up, \#AB1dn, \#CD1up, \#CD1dn, \#cd - up, \#cd - dn, \#Netdn \\ & = 0\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ 0 \end{aligned}$$

In the basic Petri net, a transition is enabled if each input place has as many (or greater) number of tokens than the input-arc cardinality. An SPNP specification allows in addition, an *enabling function* for each transition. Thus, in SPNP, a transition needs the requisite number of tokens in each input place *and* the enabling function to evaluate to *True*, in order to be enabled. Therefore, in SPNP enabling conditions can be moved to the enabling function to reduce clutter in the graph. The enabling function capability can also be used to decompose Petri nets having

unwieldy pictorial descriptions. We shall illustrate this decomposition in Section 5 (see Figure 10). For transitions enabled exclusively by an enabling function (i.e., transitions with no input arcs) the burden of disabling the transition after it had fired falls on the programmer and can result in complicating the enabling function. In contrast, an explicit input arc from the *enabling place* can automatically disable the transition by a routine movement of tokens at firing time. Clearly, a balanced approach is best. If two timed-transitions in an SRN are enabled simultaneously, the transition with the minimum firing delay fires first. Immediate transitions have priority over timed transitions. Simultaneously enabled immediate transitions with distinct enabling conditions and having the same priority fire non-deterministically (as after the firing of *Tbth* in Figure 5). The competition between immediate transitions having the same priority and a common enabling condition is resolved probabilistically. SPNP provides for the specification of probabilities and priorities. Details of SPNP are available in [11].

We now present the method for automatic generation of the state space and state transitions of the Markov chain underlying an underlying specification. Recall that the Petri net executes through token activity at enabled transitions. Assume that the initial marking satisfies the enabling conditions for one or more transitions. (In Figure 5 transitions *T-AB1fl*, *T-CD1fl* and *T-bth* are enabled). A transition fires, (possibly) yielding another distinct marking. The firing causes some transitions to be disabled and new ones to be enabled. The set of distinct markings together with the transitions is called the *reachability graph* of the Petri net. In Figure 6, we show the reachability graph and underlying Markov chain for the Petri net in Figure 5. The reachability graph for an SRN differentiates between markings which enable immediate transitions and those which do not enable immediate transitions. The former are called *vanishing markings*, the latter, *tangible markings* [30]. In Figure 6, tangible markings are shown by solid boxes and vanishing markings by dotted boxes. (The original Petri net does not address the issue of transition time.) The set of tangible markings is the state space of the underlying Markov chain. To construct the Markov chain state transition diagram, vanishing markings in the reachability graph are reduced to a *point* and used as a random switch with suitable probabilities. These probabilities appear in the Markov chain specification. In the example *T-ABnwfl* and *T-CDnwfl* can be simultaneously enabled, but have different enabling functions. Equal priority is assumed (Figure 6).

If a vanishing marking enables both timed and immediate transitions simultaneously, the timed transition will never occur because of lower priority and the arc to the timed-transition may be removed. The time spent in a tangible marking is the time between the instant it is entered and the instant that the first enabled transition fires. The time-to-firing is exponentially distributed with

parameter equal to the sum of parameters of enabled transitions in the marking – for the initial marking in Figure 5, a sample is drawn from an exponential distribution with rate $(\lambda_{T-AB1fl} + \lambda_{T-CD1fl} + \lambda_{Tbth})$.

The previous paragraph indicates that the underlying Markov chain can be obtained through a systematic, relatively easy modification of the reachability graph. After generation, the Markov chain solution is essentially a solution of a system of ordinary differential equations. For Petri nets representing non-Markovian processes, the underlying process-description bears little or no resemblance to the reachability graph and generation methods are also different. Markov regenerative stochastic Petri nets (MRSPN) [9] and Deterministic Stochastic Petri nets (DSPN) [28] can be used to specify non-Markovian processes. Relative to Markov chain methods, non-Markovian solution methods are complex [12].

The Markov reward model is derived from a Markov chain by assigning a *reward rate* to each state. For many measures of interest, the reward process is completely specified by the set of reward rates and the SPN. The combined specification is the Stochastic Reward Net (SRN) which SPNP takes as input. In this section, we discuss the relationship between the reward process and the Markov chain and sketch the procedure for obtaining measures of interest from an SRN specification.

Many measures of interest such as reliability, interval availability, instantaneous availability, response time distribution, are expressible as the *expected value* of the random variables describing the reward process. Let $\{Z(t), t \geq 0\}$ be the family of random variables describing the Markov chain underlying the SRN. For each $t \geq 0$, the random variable $Z(t)$ takes values in T , the set of tangible markings. Assign a reward rate r_i to each $i \in T$. For a fixed t , if $Z(t) = i$, then, $X(t) = r_i$. Then, the set of distinct reward rates R , is the state space for the reward process $\{X(t), t \geq 0\}$. The expected value of $X(t)$ is:

$$E[X(t)] = \sum_{r \in R} r P\{X(t) = r\}.$$

Since both $X(t)$ and $Z(t)$ are defined on the same probability space,

$$\{X(t) = r\} \leftrightarrow \bigcup_{i \in T \wedge r_i = r} \{Z(t) = i\}$$

and

$$P\{X(t) = r\} = \sum_{i \in T \wedge r_i = r} P\{Z(t) = i\}.$$

Hence,

$$E[X(t)] = \sum_{i \in T} r_i P\{Z(t) = i\}.$$

Thus, the expected value of random variable $X(t)$ is a function of the solution to the underlying Markov chain and the set of reward rates (reward accumulated in a state is the product reward rate and the sojourn time in that state). Clearly, the information contained in an SRN specification is sufficient for computing any measure expressible as a linear combination of an underlying measure of the Markov chain. This underlying measure is defined by the primary measure of interest. In the discussion above, we assumed that the measure of interest, $E[X(t)]$, is a linear combination of the time-varying probability of occupation of the states of the Markov chain. Reliability, $R(t)$ is a time-varying measure obtained as a linear combination of $P\{Z(t) = i\}$. A reward rate of 1 for up-states and 0 for down-states accumulates the time-varying probabilities of the up-states. The reward rate specification is at the net-level and is translated by the tool into assignments to individual states in the underlying Markov chain. In summary, to obtain a given measure using SPNP, it is necessary to define this measure in terms of an underlying measure (for which to solve the Markov chain) and to assign appropriate reward rates.

A Markov chain can be solved with respect to several measures, time-varying probability of state occurrence $P\{Z(t) = i\}$ and the steady-state probability of state occurrence $\lim_{t \rightarrow \infty} P\{Z(t) = i\}$, being the most common ones. The former consists of solving a system of ordinary differential equations, the latter, a system of linear algebraic equations. Another underlying measure is *the expected time spent in a state within an interval $[0, t]$* given by

$$\int_0^t P\{Z(x) = i\} dx.$$

This, again, requires the solution of a system of ordinary differential equations. (It is not necessary to obtain $P\{Z(t) = i\}$ and then integrate [40]). A linear combination using the specified reward rates yields the expected accumulated reward in the interval $[0, t]$; for $t \rightarrow \infty$, it gives the expected accumulated reward until the time-to-absorption. To see this, note that $E[X(t)]$ is the expected instantaneous reward rate so that the expected *accumulated* reward over an interval $[0, t]$ is:

$$E[Y(t)] = \int_0^t E[X(x)] dx.$$

Substituting for $E[X(x)]$ from the earlier derivation,

$$E[Y(t)] = \sum_{i \in T} r_i \int_0^t P\{Z(x) = i\} dx.$$

The computation of the underlying measures using numerical techniques is described in [10]. SPNP allows for the specification (and solution) of a variety of time-varying and steady-state measures. A discussion on performability measures is available in [31] [40].

4 Network Architecture and Alternate-Routing

Corporations and businesses often build their networks by leasing bandwidth from a telecommunications company to interconnect their office locations. They are free to use the bandwidth along the leased lines in any way they want. In this section we outline the architecture of B-ISDN and discuss the relevance of the graph model we have used in Section 2. We briefly describe our model of network operation and introduce three protocols for alternate-routing.

4.1 Network Architecture

We consider a layered network architecture where the lowest layer is a facility network of transmission facilities and switches called cross-connects. If two nodes which are not adjacent to each other need to communicate, a *path* is established by configuring the cross-connects. Cross-connects may be configured to provide more than one path between two nodes. The set of paths between any two nodes is called a *path group* [23].

Corporate network owners introduce a second level of switches called *call switches* to switch circuits along leased lines and to manage their network. Leased lines are simply path groups connecting office locations. *Circuits* carry application-specific information along paths and are termed *physical* or *virtual* depending on the multiplexing policy. For B-ISDN, circuits are virtual and paths are called *virtual paths*. The edges of the graph used to represent a network in our fault-tree examples represent path groups of the corporate network while nodes represent call-switches placed at office locations.

Circuits connecting call switches are either *single-hop links*, i.e., having no intermediate call switches, or *two-hop links* or *vias* with two circuit-links tied together at a call switch. The direct or single-hop link is usually the primary route while alternate routing can be provided by *vias*. Alternate routing through concatenation of more than two links is avoided due to network stability considerations. For details on the network architecture see [23] and for stability issues, [20].

Our objective is to assess the reliability of a *corporate network application* which requires a given set of location-pairs to be continuously connected by circuits over a specified time interval. We refer to the set of location-pairs as the *connection pattern*; one may view this set as a generalized *call*. In defining the network operation, we assume that each connection requires exactly the same bandwidth. Also, each connection is realized by a single-hop link at *call set-up* and the connection pattern remains unaltered for a given call session. Alternate connections are restricted to vias. Each link of the via connection is called a *via link*. In the following section we present three alternate routing protocols.

4.2 Alternate-Routing Protocols

We discuss the mechanisms of failure, reconfiguration and repair through the example in Figure 7. The figure shows three location-pairs connected by links 1, 2 and 3. (Three single-hop connections are shown, not one 3-hop connection.) In the figure, solid lines are active links, dotted lines are free links, and dashed lines are reserved or failed primary links. An active link carries traffic for the application of interest. Free links are links that can potentially carry this traffic along alternate paths.

When a primary link fails (that is, no path in the associated path group is able to provide a circuit), the single-hop connection is replaced by a via by tying two free links at the call switch. Available vias have relative priorities. Free links may be *up* or *down*. We assume that if an alternate connection is available, the network recovers perfectly, i.e., that the switching mechanism never fails when an alternate route is available. Although we do not include reconfiguration processes internal to the switch here, the effect of the switch can be modeled through the coverage parameter ³ [18] because the switch-reconfiguration process operates on time scales which are typically six orders of magnitude smaller than that of failure processes. This is an approximation exploiting the relative time-scales of participating processes.

To continue our description of network operation, failed vias cannot be re-routed over vias; if a via fails before the corresponding primary-route link is repaired, the network does not seek out alternate vias. This is an assumption which greatly simplifies modeling. However, we do not expect it to have a serious impact on the results for the following reason. To achieve a reasonable level of overall network reliability, repair rates need to be significantly higher (by three orders of

³The coverage parameter is the probability of the system's recovery given that a failure has occurred [7]. A sub-model or fault-injection experiments are often used to obtain this parameter.

magnitude) than the failure rate of a via. So, at any time, the probability of occurrence of several failed alternate routes is significantly reduced by the repair process [1].

An issue relevant to alternate-routing is the bandwidth (the combined capacity of underlying paths) of links. Bandwidths are non-zero and finite. However, in our models, we shall consider two extreme cases: the unit capacity case and the infinite capacity case. The rationale for this is to identify the approximate range of values over which the reliability can vary. Two protocols, called *Non-restoring on Repair* (NROR) and *Restoring on Repair* (ROR), respectively, differ in their responses to repair of the primary single-hop link route. These protocols assume unit capacity links and are described below. The third assumes infinite capacity links and serves as a bound for the NROR and ROR protocols. Additional features of this protocol are presented below under *A Third Protocol*.

NROR and ROR Protocols: We first consider the NROR protocol (see Figure 7). At call set-up, all connections are made through primary-routes over single-hop links. When a primary-route link fails, the network maintains the connection by transferring traffic to the available via with the highest priority. If no vias are available, the network fails. When a link is repaired, it becomes available for providing a connection for the corporate network. The NROR protocol does not restore traffic to the primary route upon its repair. Instead, it places the repaired primary-route link in a *reserved* state (shown by dashed lines in Figure 7). Only upon a via link failure is this corresponding reserved single-hop link used, i.e., the repaired single-hop link is *reserved* to carry the traffic of one specific location-pair in the connection pattern. If the primary link is down when the via fails, the network fails. Intuitively, the NROR protocol is an example of a poor policy because it is wasteful of resources.

The ROR protocol, in contrast, instantly restores traffic to the primary route upon its repair. Intuitively, this is a good routing policy. The responses of the NROR and ROR protocols to primary-route repair are shown in Figure 8.

A Third Protocol: Here we construct a protocol for comparing the effect of traffic restoration on repair, of providing infinite capacity, and of exhaustively seeking out alternate vias for re-routing, on the network-application's reliability. The construction is as follows. First, we allow link capacities to be infinite so that if a link is *up*, it is guaranteed to be available as an alternate route. In the NROR and ROR protocols, a link may be *up* and yet not be available if capacity is used up. Second, we allow vias to replace vias (in NROR and ROR, traffic must alternate between a direct link and its set of vias). Our goal is to compare this protocol with the ROR protocol, which

we expect will have higher reliability than the NROR protocol – the NROR must serve as a lower bound in a relative sense. We seek to rank the three protocols through a reliability evaluation. Intuitively, the infinite link capacity protocol is ranked the best, followed by the ROR protocol.

For the three protocols, given that the network application is initially set-up, it is considered to fail at the instant when a connection of the generalized-call (connection-pattern) fails and this connection cannot be re-established through an alternate route. In the next section, we specify and solve Petri net reliability models for the three protocols, using SPNP.

5 Specification through Petri Nets

In the previous section we described the failure, repair and reconfiguration mechanisms for a corporate network application by means of diagrams (Figure 7 and Figure 8) and through a textual description of the alternate-routing protocols. In the figure we used solid, dashed and dotted lines to convey whether a link was in use, reserved, or not in use. Further, by means of an **X** we showed whether links were up or down. Details of the protocol not apparent from the diagram were described in the text and by labels/legends on figures. While this medium of conveying information might be acceptable, it has several drawbacks. In the next two paragraphs we discuss some of the drawbacks that a Petri net representation can overcome.

First, the description of the protocol in the previous section is very likely to be imprecise, making it open to interpretation. Such interpretation can introduce unintended features or lead to contradictions in the specification. A formal specification, i.e., a medium for specifying and interpreting the specification in exactly one way is required. The Petri net is such a medium for formal specification of discrete event systems with a convenient graphical representation. (The graphical representation may be used if desired.) Specification in a high-level Petri net language such as CSPL can be compiled into basic Petri net constructs. This allows a user to develop a formal specification of a discrete event system with relative ease. As mentioned earlier, the Petri net *graph* is like a flowchart for guiding the development of a program for translation into some convenient (high-level) Petri net language. The high-level language in turn, brings to the user the power and flexibility of expression offered by high-level programming languages.

The second problem with the protocol specification in the previous section is the use of the *state space* to convey the mechanics of the protocol. Indeed, Figure 7 and Figure 8 showing a

partial state space and state transitions caused by discrete events, are the (partial) state space of a Markov chain if exponentially distributed time-to-link-failure and time-for-link-repair are assumed. Recall that this state space is identical to the reachability graph of a Petri net suitably processed to remove vanishing markings. The information contained in the vanishing markings corresponds to the information contained in the *text* of the previous section. It is clear then, that the alternate-routing protocols verbosely and ambiguously described in the previous section can succinctly and precisely be specified by a Petri net, and in a form ready for automation (Figure 10). In the next section we present a Petri net specification for the three protocols by means of Petri net *graphs*. We translated these into CSPL, and specified reward rates to obtain application-reliability using SPNP.

While Petri nets offer the conveniences described above, they explicitly generate the state space. For many real-life problems, problem sizes may be small enough (100,000 states) for a stochastic Petri net processor to exhaustively generate the state space. For larger problem sizes, appropriate methods of state space management must be incorporated into the Petri net as part of the specification [12]. This is an area of on-going research. The handling of large state spaces at the Markov chain level for protocols similar to the ones in this paper is addressed in [4] [3].

5.1 Specification of Alternate-Routing Protocols

Let us denote the total number of links in the corporate network by N and the number of primary (direct) links in the connection pattern by L . For the example shown in Figure 9, $N = 10$, $L = 3$. The primary single-hop routes are labeled from 0 to 2. The seven potential via links are numbered from 0 to 6. Figure 10 shows the subnets of the stochastic Petri net representing both the NROR and ROR. The differences between NROR and ROR can be incorporated into the enabling functions associated with some transitions. Hence, the same pictorial representation serves for both protocol specifications.

First, consider the NROR protocol. The subnet shown in Figure 10(a) describes the failure-and-repair behavior of a primary link. Two (timed) transitions correspond to the failure and repair events. λ and μ are parameters for exponentially distributed time-to-failure and time-for-repair, respectively. A token in place *direct-failed* enables the repair transition *drepair*. The presence of this token also inhibits the firing of transition *dfailure*. The *inhibitor arc* is denoted by a bubble instead of an arrowhead and enables a transition only if the number of tokens in the input place

is strictly less than the input arc-cardinality. Since there is no token in place *direct-failed*, in the initial marking, the failure transition is enabled at time zero, but not the repair transition. The initial marking contains a token in *inuse-direct* (subnet (c)) for each of the L direct links and corresponds to the L direct connections at call set-up. After an exponentially distributed time interval, a failure transition will fire. By default, the cardinality of the arcs is 1 and so the firing of *dfailure* causes one token to be generated and deposited in *direct-failed*. This firing disables transition *dfailure* and enables *drepair*. Upon repair completion, which occurs after an exponentially distributed time following token arrival, *drepair* fires and flushes the token. The parameters of the exponentially distributed times-to-firing in a marking are the sum of the parameters of the enabled timed-transition distributions. We have used L copies of subnet (a) to represent the L direct links. Similarly, subnet (b) describes the failure-and-repair cycle of the $(N - L)$ potential via links in the network. Note that these two subnets do not directly interact with each other or with any other subnet in the Petri net specification. The graphical representation is not unique and the behavior may be represented, for example, with two places representing the up and down states of the link and using two timed-transitions with four arcs; a token circulates between the up and down places. In Figures 10(a,b), we have chosen to represent the up/down status by the absence/presence of a token in place *direct-failed* and *vlink-failed*.

According to the NROR protocol, if a direct link is up, it may or may not be in use, for repaired direct links are reserved (see Figure 8). So we introduce another place *inuse-direct* (subnet (c)). Note that the response of the network to failure of a direct link *in use* is different from failure of a link in a reserved status. In the former case, a reconfiguration is initiated to transfer the traffic and in addition the direct link goes into repair. In contrast, a failed reserved link merely goes into repair. Since in practice, the time for reconfiguration is small relative to time-to-failure and time-for-repair, reconfiguration can be assumed to occur instantaneously. Accordingly, in subnet (c) (L instances of which correspond to the L primary direct connections), if there is a token in place *inuse-direct* and the direct link fails (the failure being registered in the corresponding instance of subnet (a) by the arrival of a token), immediate transition *pass-on* fires. This triggers off a sequence of immediate-transition firings resulting in a tangible marking which has a token each in place *inuse-alt* (subnet (e)), of the selected via links for an alternate path. This sequence of events is discussed below. Note the $(N - L)$ instances of subnet (e).

Transition *pass-on* fires if there is a token in *inuse-direct* and the associated marking-dependent enabling function evaluates to *True*; we use the generic symbol $\langle efn \rangle$ to indicate that there is an associated enabling function. The functions are summarized in Table 1. For transition *pass-*

Transition	Enabling condition/ Priority / Arc multiplicity
Subnet (c), $i \in \{0, 1, 2\}$	
pass-on.i flush-reconfg.i takeback.i full-cycle.i pass-on.i:moarc.i full-cycle.i:miarc.i	$\#direct-failed.i:net-a == 1$ $\#selector:net-d == \#myaddr.i:net-c$ <i>Priority</i> > <i>pr1-trans:net-d</i> , <i>pr2-trans:net-d</i> $(\#retaddr.*:net-e == \#myaddr.i:net-c) \wedge (\#restore.*:net-e == 1)$ $\wedge (\#inuse-direct:net-c == 0) \wedge (\#direct-failed.i.*:net-a == 0)$ $\#inuse-direct.i:net.c == 1$ <i>i</i> +1 (<i>i</i> = 0 will permanently enable transition. So add 1.) <i>i</i> +1
Subnet (d)	
beg-reconfg.i pr1-trans pr2-trans beg-reconfg.i:moarc.i pr1-trans:viarc pr1-trans:voarc pr2-trans:viarc pr2-trans:voarc	$\#reconfg.i:net-c == 1$ (alt. route 1-2 UP) $\vee$ (alt. route 2-4 UP) (alt. route 0-6 UP) $\vee$ (alt. route 0-5 UP) $\vee$ (alt. route 3-5 UP) <i>Priority</i> > <i>pr1-trans:net-d</i> <i>i</i> +1 $\#selector:net-d^\dagger$ $\#selector:net-d + Bias$ (<i>Bias</i> > 0 keeps takeover enabled for two firings) $\#selector:net-d^\dagger$ $\#selector:net-d + Bias$
Subnet (e), $j \in \{0 \dots 6\}$	
takeover.j, j = 0,3,5,6 takeover.j, j = 1,2,4 giveback.j endrestr.j twinflush.j takeover.j:viarc, j = 0,3 takeover.j:viarc, j = 2,5,6 takeover.j:viarc, j = 1,4 takeover.j:voarc, j = 2,5,6 free-via.j:viarc	<i>via-link j</i> is in alt. route for failed primary link $\wedge$ alt. route is UP Sequencing Priority of <i>j</i> = 0,3 > <i>j</i> = 5,6 <i>via-link j</i> is in alt. route for failed $\wedge$ alt. route is UP $\wedge$ higher priority alt. route is DOWN Sequencing Priority of <i>j</i> = 1,4 > <i>j</i> = 2 $\#vlink-failed.j:net-b == 1$ $\#inuse-direct.(\#return-addr.j):net-c == 1$ $\#return-addr.j:net-d == \#return-addr.(twin-of-j):net-d$ $\wedge \#restore.(twin-of-j):net-d == 1$ <i>Priority</i> > <i>takeback.(\#return-addr.j):net-c</i> $\#pr2-trans:net-d - Bias^\dagger$. Also for voarc <i>Bias</i> † . (These links have lower sequencing priority.) $\#pr1-trans:net-d - Bias^\dagger$. Also for voarc $\#retaddr.twin-of-j:net-e$ $\#return-addr.j:net-e^\dagger$

† If input place is empty, return 1.

Table 1: Summary of enabling functions, transition priorities and arc multiplicities.

on, the marking-dependent enabling function *returns True* if there is a token in *direct-failed* of the corresponding (a)-subnet; otherwise, it returns *False*. This function can be coded in C-based CSPL. The firing of *pass-on* deposits a number of tokens equal to the address of the failed primary link in *myaddr* in subnet (c); $\langle moarc \rangle$ on the output arc of *pass-on* denotes multiple cardinality of the output arc ($\langle miarc \rangle$ is used to denote the multiplicity of input arcs. See Table 1). Address values 1 to L respectively, are assigned to $\langle moarc \rangle$ of the L instances of subnet (a). The firing of *pass-on* also generates and deposits a token in *reconfg* (and consumes the token in *inuse-direct*).

Since immediate transitions take priority over timed transitions, all immediate firings will be completed before any timed transition fires. Thus, there will never be two simultaneous reconfigurations. With a token in *reconfg*, the switch shown in subnet (d) which selects the alternate path, is activated. This could have been done by connecting place *reconfg* of each of the L subnets (a) to place *selector* in subnet (d). However, here we demonstrate the use of a transition enabled exclusively by an enabling function — this feature allows us to decompose a large Petri net into subnets through the elimination of explicit arcs. So, L transitions labeled *beg-reconfg* and having no input-enabling places are connected to *selector*. Each is enabled by a marking-dependent enabling function (see Table 1) which checks the status of place *reconfg* in the corresponding subnet (c). Thus, arrival of a token in *reconfg* causes the corresponding *beg-reconfg* transition to fire and deposit a number of tokens equal to the address of the primary link requesting reconfiguration into *selector*. (Accordingly, $\langle moarc \rangle$ appears on the output arcs of *beg-reconfg*.) But once firing has occurred, an explicit disabling of *beg-reconfg* must be done. This is achieved by introducing an immediate transition *flush-reconfg* (subnet (c)) which fires only *after* *beg-reconfg* has fired. The sequencing of firings is achieved through an enabling function for *flush-reconfg* which checks for token-presence in *selector*; however, this is not enough, because tokens in *selector* enable two other immediate transitions *pr1-trans* and *pr2-trans*. According to the rules in Section 3, the competition between immediate transitions is resolved non-deterministically unless overridden by priority assignments. So we assign a higher priority to *flush-reconfg* relative to the *pr1-trans* and *pr2-trans* transitions to ensure that *beg-reconfg* is first disabled. The additional programming effort caused by the decomposition could have been avoided by the use of explicit arcs, but for large Petri net representations, this trade-off is often worthwhile. In the following paragraph, we consider the route selector switch (subnet (d)).

In the present set-up, alternate two-hop routes for each primary route are prioritized. In the interest of simplicity, we assume that a given via link has a fixed priority and that this link serves in an alternate path of that fixed priority, no matter which primary route it serves. Essentially, this

simplification avoids priority re-assignments to via links on a primary-route dependent basis. With this restriction, we construct the switch in subnet (d). In this example, there is a maximum of two alternate paths for any primary route and so via links have priority HIGH or priority LOW. For the example in Figure 9, via links 0,3,5,6 (and alternate paths formed using these links) have higher priority than via links 1,2,4. With one or more tokens in place *selector*, immediate transitions *pr2-trans* and *pr1-trans* are enabled by their respective enabling functions if the alternate routes of the primary link whose identity is in *selector* are available; alternate routes are unavailable if component via links are failed or (at the instant of the current reconfiguration) in use as routes for other primary links. If both routes (each with a distinct priority) are available, the higher priority route is selected and tokens deposited in the *inuse-alt* of the corresponding via links. The selection takes place in two steps: *pr2-trans/pr1-trans* is enabled under all conditions that the associated set of *takeover* transitions is enabled (subnets (d) and (e)); however, only two *takeover* transitions, corresponding to the two available via links of the selected alternate route are enabled. The alternate route-priority functions are embedded in the enabling functions for *pr2-trans/pr1-trans* and *takeover*. (We found it convenient to embed the route priorities directly into the enabling functions; these functions are therefore complex.) Also, note that a *sequencing priority* must be assigned to the two enabled *takeover* transitions or else, one will be switched in arbitrarily and treated in isolation. Thus, for each pair of links of the chosen alternate route, a sequencing priority (*SEQ-H* or *SEQ-L*) is specified by assigning (relative, pairwise) priorities to transition *takeover* of the two via-links selected for the alternate route (Table 1), with the further assurance that no other immediate transitions can fire in between. In this way, we switch two links one after the other. For the example in Figure 9, via links 0,1,3,4 have higher sequencing priority than 2,5,6 and will be switched in first. Note that the sequencing priority of a via link is not the same as the priority of the via it serves in. After the firing of the *takeover* transitions, the Petri net reaches a tangible marking (with all immediate transitions disabled and further behavior controlled by timed transitions only). We chose not to decompose subnets (d) and (e) in order to keep the enabling functions at *takeover* manageable: there are explicit arcs from *pr1-pl* and *pr2-pl* to the 7 subnets representing the 7 via links of Figure 9.

We complete our description of the switch with an explanation of the $\langle \textit{viarc} \rangle$ and $\langle \textit{voarc} \rangle$ labels. These denote variable-cardinality arcs and allow a marking-dependent number of tokens to be consumed or generated, respectively; $\langle \textit{moarc} \rangle$, $\langle \textit{miarc} \rangle$ denote *fixed* cardinality. We have used the variable cardinality arc feature to pass on the identity of the primary route to the via links in an alternate route. Accordingly, the address is stored in *return-addr* of the via links. This

completes the specification of the reconfiguration mechanism upon failure of a primary route. In the case that a reconfiguration request occurs when there are no available alternate routes, (according to the NROR protocol) the network fails. This condition is detected by a built in CSPL routine, *halting condition* which registers that this marking is a terminating or absorbing marking. Next we briefly describe traffic restoration.

When traffic is carried on an alternate route, *inuse-alt* of the component via links contain a token each, and each has a copy of the primary route address it is currently serving in *return-addr* (subnet (e)). In the NROR protocol, restoration occurs with the failure of a via link. Thus, transition *giveback* in subnet (e) fires when the corresponding via link fails (registered by a token in subnet(b)) and the marking dependent enabling function of *giveback* detects a token in *vfailure*; as a result, a token is deposited in *restore* (subnet (e)). This action triggers off another sequence of immediate transitions. The status of the primary route is checked by the enabling function of *takeback* in subnet (c). If up, but not in use, it further scans for requests of restoration from its alternate routes making use of the identity in *return-addr* — a token in *restore* causes *takeback* to fire. Once again, due to decomposition, *takeback* must be explicitly disabled through the firing of *endrestr* in subnet (e) in response to a token in *inuse-direct. (#return-addr)* of subnet (c). After restoration, transition *free-via* fires, removing all tokens from *return-addr* and *clean-up*, completing thereby the transfer of traffic from the point of view of the failed via link. But, the *twin* which is not failed must be freed as well. In fact, releasing of the twin is the first event to occur when an active via link fails, through the firing of *twinflush* with a relatively high priority with respect to *takeback. (#return-addr)*: the presence of a token in *restore* of a via-link causes *twinflush* in the subnet of its twin to fire. According to the NROR protocol, if the primary link is down at the time of request for restoration, the network fails. As in the case of reconfiguration, this condition is coded into the *halting condition*. *myaddr* in subnet (c) is flushed out to complete the reconfiguration-restoration cycle. This concludes our description of the NROR Petri net representation.

Since the ROR protocol only differs in its response to primary link repair, the repair of the primary link must trigger restoration. This is achieved by modifying the enabling function of *giveback* so that it fires on repair of a primary link rather than the failure of a via link. This simple modification, together with a change in the *halting condition*, is the SRN specification of the ROR protocol.

A *Third Protocol* introduced in Section 4.2 has infinite capacity links. This implies that primary links are never denied the alternate routes unless associated via links are down. As far as routing

is concerned, alternate routes are exhaustively sought. Under these assumptions, there is no interaction whatsoever between the subnets and the Petri net is simply N copies of subnet (a)/(b) in Figure 10. Indeed, the introduction of finite-capacity links considerably complicates the model by causing the availability of alternate routes to depend on the status of every other connection at the reconfiguring instant. The model type for the third protocol is of the type discussed in Example 3 in Section 2.

Summary of Results: The three Petri net specifications for the example corporate network shown in Figure 9 were encoded in CSPL and solved using SPNP. The unreliability graphs are shown in Figure 11. Unreliability is plotted over a four-day period. The graphs confirm our intuition on the three routing policies. It is interesting to note that restoration on repair has a significant impact on the network-application’s unreliability. The values of link-failure rate and link-repair rates used were 0.0005 failures per hour and 0.4995 repairs per hour, respectively. For the example in Figure 9, the Mean-Time-to-Failure (MTTF) of the network for the NROR and ROR policies were one-two-hundredth and one-fifth respectively, of the MTTF of the infinite link-capacity version which seeks out all available alternate paths. The MTTF values obtained for NROR, ROR and the infinite link capacity case were 4954, 7328 and 1001277 hours, respectively.

The number of tangible and vanishing markings and the number of marking-to-marking transitions for the NROR, ROR and infinite capacity protocols are respectively given by the following triples: [8280; 2136; 15000], [3808; 1332; 7328], [1024; 0; 7680]. The number of vanishing markings and the corresponding marking-to-marking transition counts vary with the Petri net specification. As a rule-of-the-thumb, the number of vanishing markings increases with the number of immediate transitions and the number of simultaneously enabled immediate transitions. A discussion of the relative merits (measured in terms of computational complexity) of eliminating and preserving vanishing markings is available in [15].

6 Conclusions

In this paper, we have developed several reliability models for a corporate network with an alternate routing capability. The simplest versions were developed using fault-trees. For more complex routing protocols and the modeling of link repair, Petri nets and Markov reward models were used. We demonstrated the use of stochastic reward nets to succinctly specify, automatically generate and solve for application-reliability under three alternate-routing protocols. While the methodology is

applicable to all networks and protocols in general, the limit of its applicability is decided by the size of the underlying state space. The current limit is about 300,000 states for a workstation and inclusive of tangible and vanishing markings. The number of vanishing markings can be reduced by careful use of immediate transitions. We recommend that the user estimate the size of the state space before building the Petri nets.

Application of the stochastic reward net methodology for reliability prediction can encounter several difficulties. First, for large networks, the underlying continuous-time Markov chain can become prohibitively large. A number of techniques such as lumping [25], truncation [24], decomposition/fixed-point iteration [32] may be applied. The inclusion of non-zero reconfiguration time in reliability models causes the continuous-time Markov chain underlying an SRN to be numerically stiff. Several stiffness-tolerant numerical methods are applicable [37, 29]; alternately, decomposition methods may be applied to avoid stiffness [6]. Finally, for models having non-exponential distributions (for example, non-exponentially distributed time-for-repair or time-to-failure), phase-type expansions and Markov regenerative processes are possible solutions.

7 Acknowledgement

The first author thanks Ricardo Fricks (Duke University) for fruitful discussions on the Petri net programming methodology and Varsha Mainkar (Duke University) for suggesting the use of subnet (d) in Figure 10 to eliminate complex enabling functions. Both authors take this opportunity to thank the reviewers for their help in making the paper suitable to this journal.

References

- [1] J. Arlat and J. C. Laprie. Performance-Related Dependability Evaluation of Supercomputer Systems. In *Proceedings of the 13th Fault-Tolerant Computing Symposium*, 1983.
- [2] B. E. Aupperle and J. F. Meyer. State Space Generation for Degradable Systems. In *Proceedings of the 21st Fault-Tolerant Computing Symposium*, 1991.
- [3] M. Balakrishnan and A. Reibman. Characterizing a Lumping-Heuristic for a Markov Network Reliability Model. In *Proceedings of the 23rd Fault-Tolerant Computing Symposium*, 1993.

- [4] M. Balakrishnan and A. Reibman. Reliability Models for Fault-Tolerant Private Network Applications. *IEEE Transactions on Computers*, 43(9), September 1994.
- [5] M. O. Ball. Computational Complexity of Network Reliability Analysis: An overview. *IEEE Transactions on Reliability*, 35(3), August 1986.
- [6] A. Bobbio and K. S. Trivedi. An Aggregation Technique for the Transient Analysis of Stiff Markov Chains. *IEEE Transactions on Computers*, C-35(9), September 1986.
- [7] W. G. Bouricius, W. C. Carter, and P. R. Schneider. Reliability Modeling Techniques for Self-Repairing Computer Systems. In *Proceedings of 24th National Conference of the ACM*, 1969.
- [8] G. Chiola. A Software Package for the Analysis of Generalized Stochastic Petri Nets. In *Proceedings of the International Workshop on Timed Petri Nets*. IEEE Computer Society Press, July 1985.
- [9] H. Choi, V. G. Kulkarni, and K. S. Trivedi. Markov Regenerative Stochastic Petri Nets. In *Proceedings of Performance'93, Rome, Italy*, September 1993.
- [10] G. Ciardo, A. Blakemore, P. F. Chimento, J. K. Muppala, and K. S. Trivedi. Automated generation and analysis of markov reward models using stochastic reward nets. In C. Meyer and R. J. Plemmons, editors, *Linear Algebra, Markov Chains, and Queueing Models, IMA Volumes in Mathematics and its Applications*. Springer-Verlag, Heidelberg, Germany, 1992.
- [11] G. Ciardo, R. M. Fricks, J. K. Muppala, and K. S. Trivedi. *SPNP Users Manual*. Department of Electrical Engineering, Duke University, Durham, North Carolina, 27708, January 1994.
- [12] G. Ciardo, R. German, and C. Lindemann. A Characterization of the Stochastic Process Underlying a Stochastic Petri Net. In *Proceedings of the 5th International Conference on Petri Nets and Performance Models, Toulouse, France*, Los Alamitos, CA, October 1993.
- [13] G. Ciardo, R. Marie, B. Sericola, and K. S. Trivedi. Performability Analysis Using Semi-markov Reward Processes. *IEEE Transactions on Computers*, 39(10), October 1990.
- [14] G. Ciardo, J. Muppala, and K. S. Trivedi. SPNP: Stochastic Petri Net Package. In *Proceedings of the International Workshop on Timed Petri Nets*. IEEE Computer Society Press, December 1989.

- [15] G. Ciardo, J. K. Muppala, and K. S. Trivedi. On the Solution of GSPN Reward Models. *Performance Evaluation*, 12, 1991.
- [16] C. J. Colbourn. *The Combinatorics of Network Reliability*. Oxford University Press, Oxford, 1987.
- [17] J. A. Couvillion, R. Freire, R. Johnson, W. D. Obal II, M. A. Quereshi, M. Rai, W. H. Sanders, and J. E. Tvedt. Performability Modeling with Ultrasan. *IEEE Software*, 8(5), September 1991.
- [18] J. B. Dugan and K. S. Trivedi. Coverage Modeling for Dependability Analysis of Fault-Tolerant Systems. *IEEE Transactions on Computers*, 38(6), August 1989.
- [19] R. Geist, M. Smotherman, J. B. Dugan, and K. S. Trivedi. Hybrid Reliability Modeling for Fault-Tolerant Computer Systems. *Computers and Electrical Engineering*, 11(2-3):87–108, 1984.
- [20] A. Girard. *Routing and Dimensioning in Circuit-Switched Networks*. Addison-Wesley, New York, 1990.
- [21] P. J. Haase and G. S. Shedler. Stochastic Petri Net Representation of Discrete Event Simulations. *IEEE Transactions on Software Engineering*, 15(4), April 1989.
- [22] B. Haverkort and K. S. Trivedi. Discrete Event Dynamic Systems: Theory and Applications. *Specification and Generation of Markov Reward Models*, 3:219–247, 1993.
- [23] J. Y. Hui, M. B. Gursoy, N. Moayeri, and R. D. Yates. A Layered Broadband Switching Architecture with Physical and Virtual Path Configurations. *IEEE J. Selected Areas in Communications*, 9(9), December 1991.
- [24] O. C. Ibe, R. C. Howe, and K. S. Trivedi. Approximate Availability Analysis of VAXcluster System. *IEEE Transactions on Reliability*, 38(1), April 1989.
- [25] J. Kemeny and L. J. Snell. *Finite Markov Chains*. Springer-Verlag, Berlin, 1976.
- [26] V. G. Kulkarni. *Lecture Notes on Stochastic Models in Operations Research*. University of North Carolina, Chapel Hill, 1990.
- [27] V. K. P. Kumar, S. Hariri, and C. S. Raghavendra. Distributed program reliability analysis. *IEEE Transactions on Software Engineering, Special Issue on Software Reliability*, 12(1), January 1986.

- [28] C. Lindemann and R. German. Discrete Event Dynamic Systems: Theory and Applications. *Modeling Discrete Event Systems with State-dependent Deterministic Times*, 3:249–270, July 1993.
- [29] M. Malhotra, J. Muppala, and K. S. Trivedi. Stiffness-Tolerant Methods for Transient Analysis of Stiff Markov Chains. *Microelectronics and Reliability*, 34(11), 1994.
- [30] M. A. Marsan, G. Conte, and G. Balbo. A Class of Generalized Stochastic Petri nets for the Performance Evaluation of Multiprocessor Systems. *ACM Computing Surveys*, 2(2), May 1984.
- [31] J. F. Meyer. Performability: A Retrospective and some Pointers to the Future. *Performance Evaluation*, 14(3-4), 1992.
- [32] J. K. Muppala, A. S. Sathye, R. C. Howe, and K. S. Trivedi. Dependability modeling of a heterogeneous VAXcluster system using stochastic reward nets. In D. Avresky, editor, *Hardware and Software Fault Tolerance in Parallel Computing Systems*, pages 33–59. Ellis Horwood, Ltd., 1992.
- [33] V. F. Nicola. Lumping in Markov Reward Processes. In *First International Conference on the Numerical Solution of Markov Chains, IFIP W.G. 7.3*, January 1990.
- [34] J. L. Peterson. Petri Nets. *ACM Computing Surveys*, 9(3), September 1977.
- [35] S. Rai and D.P. Agrawal, editors. *Distributed Computing Network Reliability*. IEEE Computer Society Press, Los Alamitos, CA, 1990.
- [36] S. Rai, Malathi Veeraraghavan, and K. S. Trivedi. A Survey of Efficient Reliability Computation Using Disjoint Products Approach. *Networks*, to appear.
- [37] A. L. Reibman. *Transient Analysis of Large Stiff Markov Models: Numerical and Approximate Solution Techniques*. PhD thesis, Department of Computer Science, Duke University, 1987.
- [38] R. Sahner and K. S. Trivedi. A Software Tool for Learning about Stochastic Models. *IEEE Transactions on Education*, 36(1), February 1993.
- [39] W. H. Sanders and J. F. Meyer. A Performability Evaluation Tool based on Stochastic Activity Nets. In *Proceedings of the ACM-IEEE Comp. Soc. Fall Joint Computing Conf.* IEEE Computer Society Press, 1986.

- [40] K. S. Trivedi, J. K. Muppala, S. P. Woollet, and B. R. Haverkort. Composite Performance and Dependability Analysis. *Performance Evaluation*, 14(3-4), 1992.
- [41] M. Veeraraghavan and K. S. Trivedi. An Improved Algorithm for Symbolic Reliability Analysis. *IEEE Transactions on Reliability*, 40(3):347–348, August 1991.

Figure 1: **A fault-tree with no repeated events.**

Figure 2: A fault-tree with repeated events.

Figure 3: A sample path from the original underlying process.

Figure 4: A sample path from the modified process.

Figure 5: **Example showing Generalized Stochastic Petri Net constructs.**

Figure 6: **Reachability graph and underlying Markov chain for Example 2.**

Figure 7: **Failure, repair and re-routing in an NROR network.**

Figure 8: **Responses to primary-link repair in NROR and ROR networks.**

Figure 9: **An example corporate network.**

Figure 10: **SRN** representation of the **NROR** and **ROR** protocols.

Figure 11: **A comparison of unreliabilities of the three protocols.**