Stochastic Reward Nets for Reliability Prediction

Jogesh K. Muppala

Dept. of Computer Science

The Hong Kong University of

Science and Technology

Clear Water Bay

Kowloon, Hong Kong

Gianfranco Ciardo

Dept. of Computer Science

College of William and Mary

Williamsburg, VA 23187, USA

Kishor S. Trivedi*

Dept. of Electrical Engineering

Duke University

Durham, NC 27706, USA

Abstract

We describe the use of stochastic Petri nets (SPNs) and stochastic reward nets (SRNs) which are SPNs augmented with the ability to specify output measures as reward-based functions, for the evaluation of reliability for complex systems. The solution of SRNs involves generation and analysis of the corresponding Markov reward model. The use of SRNs in modeling complex systems is illustrated through several interesting examples. We mention the use of the Stochastic Petri Net Package (SPNP) for the description and solution of SRN models.

1   Introduction

Combinatorial models such as reliability block diagrams, fault trees and s-t connected networks are commonly used for system reliability and availability analysis [17, 6]. These model types allow a concise description of the system under study and can be evaluated efficiently, but they cannot represent dependencies occurring in real systems [15, 20], such as imperfect coverage, correlated failures, repair dependencies, non-zero detection/reconfiguration time, performance-reliability dependence, and phased-mission system models. State space-based models such as Markov models, on the other hand, are capable of capturing various kinds of dependencies that occur in reliability/availability models [8, 9, 18].

One major drawback of Markov models is the largeness of their state space. The sizes of these Markov chains tend to be very large for complex systems. Stochastic Petri nets (SPNs) can be used to generate the (large) underlying Markov chain automatically starting from a concise description of the system. In such cases the SPN provides a high level interface for the specification of the underlying Markov model.

In the recent years, SPNs have gained much attention as a useful modeling formalism [1, 4, 16]. They have been successfully used in the analysis of several applications [2, 11, 12]. They can easily represent concurrency, synchronization, sequencing and multiple resource possession that are characteristics of current computer systems. Several automated tools that support the evaluation of SPNs or their variants are available. These include GreatSPN [3], METASAN [16], UltraSAN [7], and SPNP [5].

Traditionally, performance analysis assumes a fault-free system. Reliability and availability analysis is carried out separately to study system behavior in the presence of component faults, disregarding the different performance levels in different configurations. Several different types of interactions and corresponding tradeoffs have prompted researchers to consider combined evaluation of performance and reliability/availability [13, 19].

Most work on the combined evaluation is based on the extension of Markov chains to Markov reward models [10], where a reward rate is attached to each state of the Markov chain. Markov reward models have the potential to reflect concurrency, contention, fault-tolerance, and degradable performance; they can be used to obtain not only program/system performance and system reliability/availability measures, but also combined measures of performance and reliability/availability [13, 19].

As discussed earlier the Markov chain is generated from a concise SPN description of the system behavior. In order to facilitate the automatic generation of the Markov reward model, it is necessary to extend the SPN description language to facilitate the specification of the the reward structure in terms of SPN entities. In other words, the SPN becomes a "SPN reward model" which can be automatically transformed into a Markov reward model. We refer to SPN augmented with the reward description as stochastic reward nets (SRNs) [4].

Steady-state analysis of SRNs is often adequate to study the performance of a system, but time-dependent behavior is sometimes of greater interest: instantaneous availability, interval availability, and reliability (for a fault-tolerant system); response time distribution of a program (for performance evaluation of software); computational availability (for a degradable system). Given the reward rate specification for a Markov reward model, the expected reward rate is computed in steady-state, while the expected reward rate at time t is an instantaneous measure of interest. The expected accumulated reward in the interval $[0, t)$ and the expected accumulated reward until absorption are the cumulative measures of interest. The mean time to absorption is a special case of the expected accumulated reward until absorption.

The Stochastic Petri Net Package (SPNP) [5] allows the specification of SRN models, the computation of steady-state, transient, cumulative, time-averaged, and "up-to-absorption" measures and sensitivities of these measures. Efficient and numerically stable algorithms employing sparse matrix techniques are used to solve the underlying CTMC.

In the following sections we give an informal description of SRN and show how it is used in modeling large systems. First, we present a simple example to illustrate some of

the structural and reward rate based constructs used in SRNs. Then, we give an informal description of SRNs.

2 Stochastic Reward Nets: A Gentle Introduction

In this section, we introduce SRNs through a simple example. Let us consider a simple multiprocessor system consisting of two dissimilar processors $P1$ and $P2$. The time to occurrence of a failure in the two processors $P1$ and $P2$ is assumed to be a random variable with the corresponding distributions being exponential with rates γ_1 and γ_2, respectively. The reliability of the two processors at time t is then expressed as $R_1(t) = e^{-\gamma_1 t}$ and $R_2(t) = e^{-\gamma_2 t}$, respectively.

We consider the system to be functioning as long as one of the two processors is functioning. If we consider the failures of the two processors to be independent, then the reliability of this system, $R_{sys}(t)$ can easily be computed as,

$$R_{sys}(t) = 1 - (1 - R_1(t))(1 - R_2(t))$$

Now suppose that, when both processors are functioning, there is also a a failure mode where both processors can fail simultaneously. The time to occurrence of this event is also assumed to be exponentially distributed with rate γ_C. Computing the reliability in this situation is more complicated, because of the failure dependency introduced between the two components. Markov models [18] can easily handle such interdependencies as mentioned earlier. The SRN in Figure 1 models the failure behavior of this system.
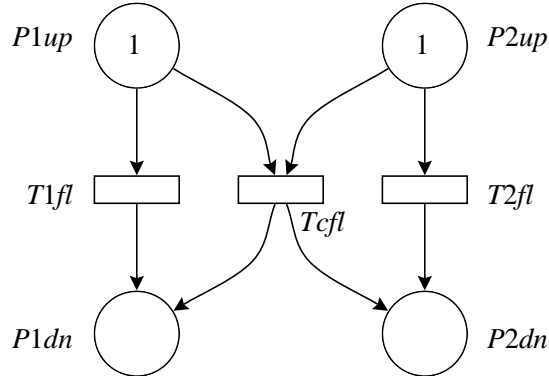

Figure 1: The SRN model of the system

In Figure 1, the main SRN constructs are shown. *Places*, shown as large circles in this figure, represent the various conditions that hold in the system. In this example, the four places, $P1up$, $P2up$, $P1dn$ and $P2dn$ represent the conditions in which the processors $P1$ and $P2$ are up or down, respectively. *Transitions*, shown in this figure as unfilled rectangles, represent the various events that could occur in the system. In this example, the three timed transitions, $T1fl$, $T2fl$ and $Tcfl$ represent the failure of the processors $P1$, $P2$ and the common-mode failure, respectively. We also notice *arcs*, shown as directional arrows,

drawn from places to transitions and from transitions to places. An arc drawn from a place to a transition is called an *input arc* into the transition from the place. Conversely, an arc drawn from a transition to a place is called an *output arc* from the transition to the place. We also notice some small filled dots that are present in the two places $P1up$ and $P2up$. These are called *tokens*. The presence of a token indicates that the corresponding condition holds. The distribution of tokens in the various places of the SRN is referred to as the *marking* of the SRN.

We consider a transition *enabled* if each of its input places contains at least one token. An enabled transition may *fire* removing a token from each of its input places and depositing a token in each of its output places. We know that the events associated with the transitions above take a period of time to happen. For example, the time to failure of $P1$ is known to be exponentially distributed with rate γ_1. This is modeled in the SRN by associating a *firing time* with each of the transitions. The firing time (which is a random variable) is the time that elapses from the time point at which the transition becomes enabled to the time point at which the transition actually fires. The firing of a transition causes the redistribution of the tokens in the SRN, perhaps resulting in a new marking.
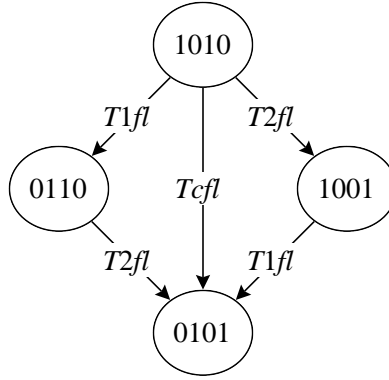

Figure 2: The reachability graph for the SRN

The set of all such markings together with the transitions among them is referred to as the *reachability graph* of the SRN. The reachability graph for the example SRN is shown in Figure 2. In this figure, each circle represents a unique marking of the SRN and the elements of the vector of 0s and 1s given within each circle is the number of tokens in places $P1up$, $P1dn$, $P2up$ and $P2dn$, respectively. The directed arrows show how the system moves from one marking to another with the firing of the appropriate transition.

This model can be specified to a Petri net based tool like SPNP [5] and solved. As an example, we compute the reliability for this system assuming that $\gamma_1 = \gamma_2 = 10^{-3}$ per hour and $\gamma_C = 5 \times 10^{-4}$ per hour. The reliability of this system as a function of time is plotted in Figure 3. This figure shows the reliability of the system with and without the common-mode failure (CMF). We can observe the contribution of the common-mode failure to the degradation in the reliability of the system. The mean time to failure (MTTF) of the system with common mode failure is 1200 hours, while the MTTF of the system without common mode failure is 1500 hours.
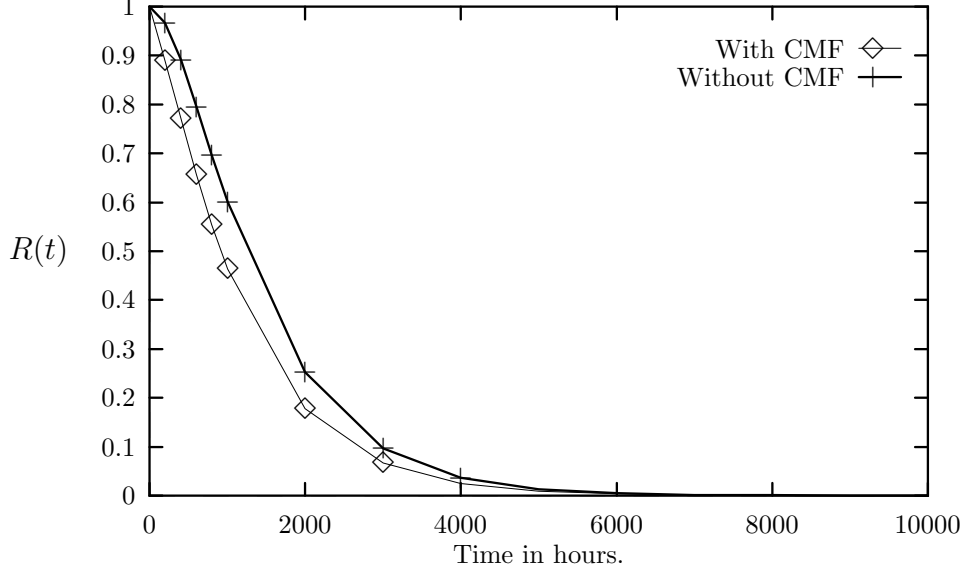
4

Figure 3: The reliability for the system

The reliability is computed by assigning appropriate *reward rates* to the states of the SRN. The reward rates are assigned based on the output measure of interest, in this case the reliability. We have already seen the various states of the SRN being represented in the reachability graph. Instead of explicitly identifying the states in the reachability graph, we can specify the reward rates associated with certain conditions of the system. Whenever a condition is satisfied in a state, the corresponding reward rate is assigned to that state.

As an example, in the computation of the reliability of the system, we wish to identify those states in which the system is functioning, and assign a reward rate of 1 to those states. We will assign a reward rate of 0 to the other states. In the SRN of Figure 1, the system is operational as long as there exists a token in either of the two places, $P1up$ or $P2up$. Thus the reward rate assignment can be specified as follows:

$$r_i = \begin{cases} 1 & \text{if } (\#(P1up) = 1) \lor (\#(P2up) = 1) \text{ in marking } i \\ 0 & \text{otherwise} \end{cases}$$

Here, r_i represents the reward rate assigned to state i of the SRN, and $\#(p)$ represents the number of tokens in place p. The reliability of the system at time t is computed as the expected instantaneous reward rate $E[X(t)]$ at time t. Expressions for the computation of the expected reward rates are given later in this paper. The MTTF of the system can also be computed using the same reward rate assignment as above.

We will introduce more complexity into the example above, by considering state-dependent failure rates as well as repair of the system. When only one processor is functioning, the rate of failure could be correspondingly altered to reflect the fact that a single processor shares the load of both the processors. When only $P1$ is functioning, we assume its failure rate is γ_1' and when only $P2$ is functioning, its failure rate is γ_2'. We could consider repair of the processors where the time to repair the processors is also exponentially distributed with rates δ_1 and δ_2, respectively. We also assume that when both the processors are waiting for repair, $P1$ has priority for repair over $P2$.

5

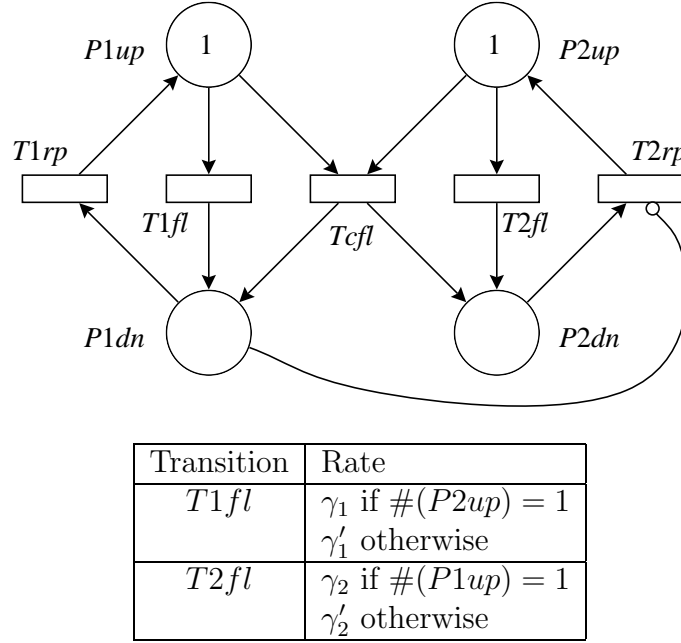| Transition	Rate
$T1fl$	γ_1 if $\#(P2up) = 1$ γ_1' otherwise
$T2fl$	γ_2 if $\#(P1up) = 1$ γ_2' otherwise

Figure 4: The SRN model of the extended system

The corresponding SRN model, shown in Figure 4, contains two additional transitions, $T1rp$ and $T2rp$, which represent the repair of the two processors. It also contains a new type of arc, from $P1dn$ to $T2rp$, with a small circle instead of an arrowhead: an *inhibitor arc*. If an inhibitor arc exists from a place to a transition, then the transition will not be enabled if the corresponding *inhibitor place* contains a token. Here, we use the inhibitor arc to give priority of repair to $P1$ over $P2$. The different failure rates of the processors are reflected by defining the firing rate of the corresponding failure transitions to be a function of the marking of the SRN.

In this case, it makes more sense to talk about the system availability than the system reliability, since we allow repair from the system failure state. We consider the system to be available as long as one of the two processors is functioning. We compute the instantaneous availability of the system (probability that the system is available at time t) as the expected instantaneous reward rate of the system with the same reward rate assignment as used for reliability. The expected interval availability, i.e., the fraction of time the system is available in the interval $[0, t)$, is given by the time-averaged expected reward rate.

The instantaneous and interval availability for the system is plotted in Figure 5. We assume $\gamma_1 = \gamma_2 = 10^{-3}\mathrm{hr}^{-1}$, $\gamma_C = 5.0 \times 10^{-4}\mathrm{hr}^{-1}$, and $\delta_1 = \delta_2 = 1\mathrm{hr}^{-1}$ As expected, the instantaneous and the interval availability decrease with time and reach a steady-state value, which is equal to the steady-state availability of the system.

We can consider one further extension to this model, by considering imperfect coverage of the failure of the processors. Whenever a processor suffers a failure, with some probability c, called the coverage probability, the failure is properly detected. With probability $1 - c$, the processor suffers an uncovered failure, wherein the failure goes undetected. We assume that this undetected failure results in the other normal processor also failing, causing the
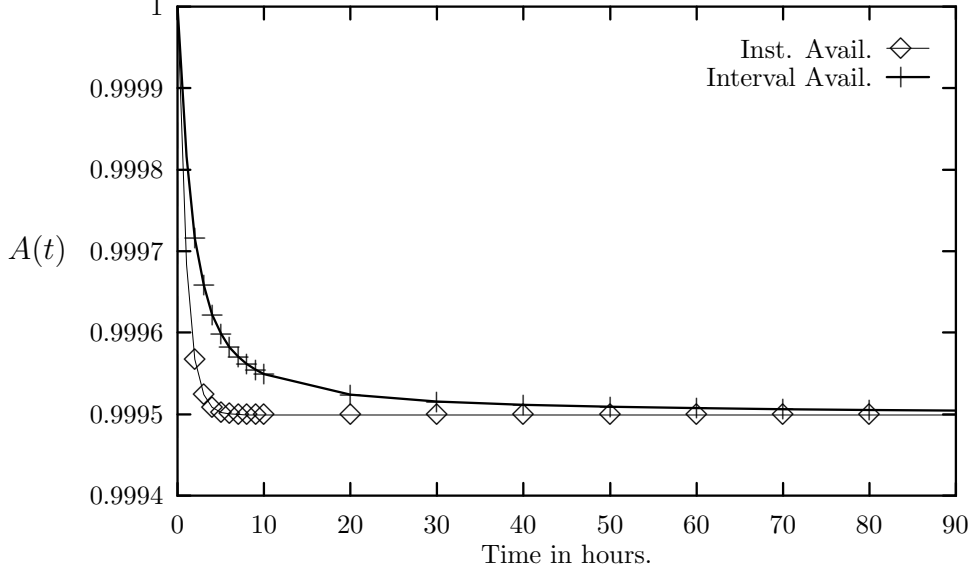
Figure 5: The availability for the system

system failure.

This scenario can also be handled by appropriately modifying the earlier SRN model of the system, resulting in the new SRN model shown in Figure 6.

In this figure, there are four new transitions, $T1cov$, $T1uc$, $T2cov$ and $T2uc$, drawn as thin bars. They are referred to as *immediate* transitions. Here, $T1cov$ and $T1uc$ represent the covered and uncovered nature of the failure of processor $P1$, respectively. The other two transitions $T2cov$ and $T2uc$ represent the same for processor $P2$. An immediate transition can be considered as the limiting case of a timed transition, when the firing rate approaches infinity. Hence, when enabled, it requires zero time to fire. We associate firing weights with these transitions. In this model, the weight associated with $T1cov$ and $T2cov$ is c, and the weight associated with $T1uc$ and $T2uc$ is $1 - c$, respectively. We also notice two additional places in the figure, namely $P1cov$ and $P2cov$. These places represent the condition under which a decision is being made whether the failure is covered or uncovered. The reachability graph for this SRN is shown in the upper portion of Figure 7, where the order of the places in the description of the markings is $P1up$, $P1cov$, $P1dn$, $P2up$, $P2cov$, and $P2dn$. The markings surrounded by a dotted line are *vanishing*, that is the SRN does not spend any time in them, since they enable immediate transitions which fire as soon as they become enabled. When analyzing the SRN, these markings can be eliminated, resulting in the reduced reachability graph shown in the lower portion of the same figure. Arcs are now labeled with sequences of transitions. For example, it is possible to go from marking 100100 (system completely functional) to marking 001001 (system down) in three ways: by firing transition $T1fl$ followed by immediate transition $T1uc$, or transition $T2fl$ followed by immediate transition $T2uc$, or transition $Tcfl$. When the labels are replaced by transition rates, the reduced reachability graph becomes the continuous time Markov chain that is solved to obtain the requested measures.

The availability of the system for different values of the coverage parameter c is plotted in

7
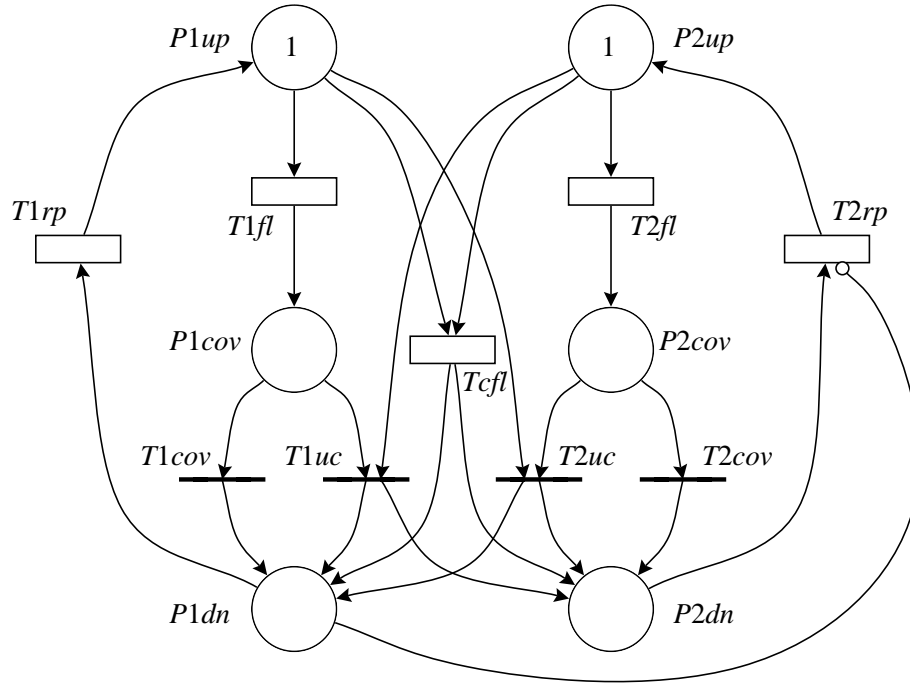
Figure 6: The SRN model of the system with covered and uncovered failures

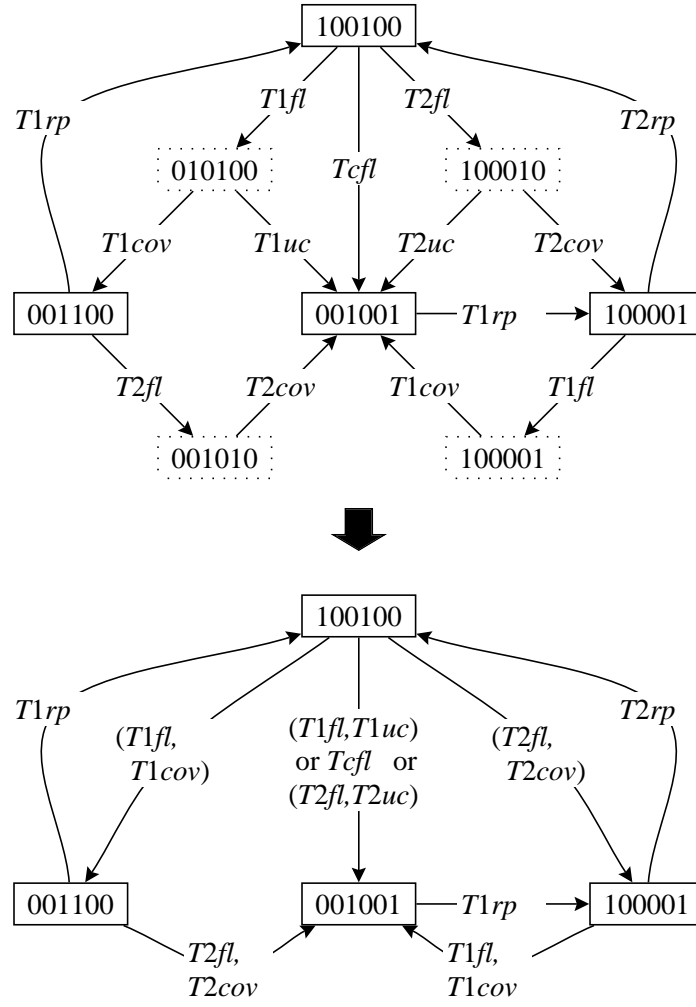| Transition	Rate
$T1fl$	γ_1 if $\#(P2up) = 1$
	γ_1' otherwise
$T2fl$	γ_2 if $\#(P1up) = 1$
	γ_2' otherwise

Figure 7: The reachability graph for the system with covered and uncovered failures
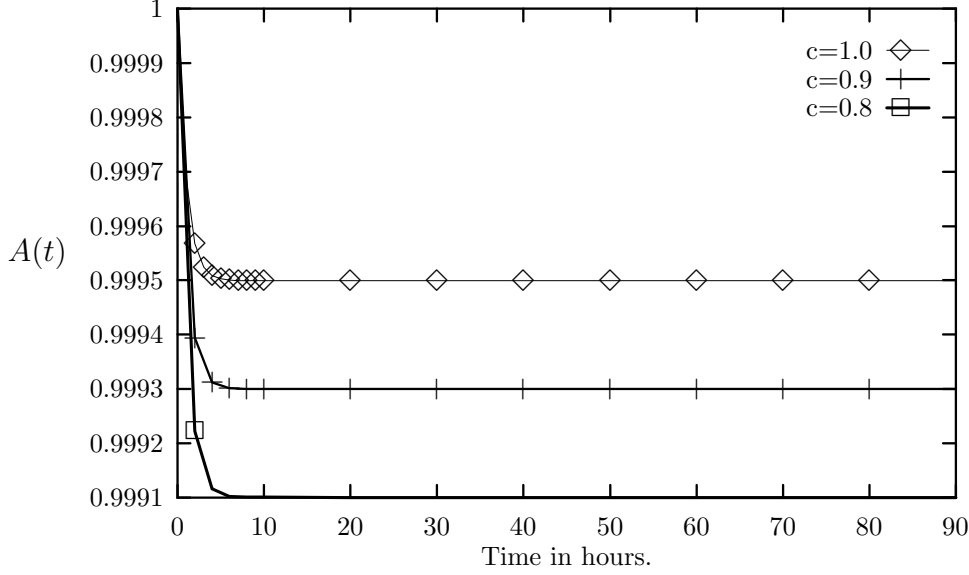
Figure 8: The availability for the system with coverage

Figure 8. From the figure we notice that with decreasing coverage probability, the availability of the system also correspondingly decreases.

3 Important features of SRNs

In this section, we give a very informal description of the features of SRNs. A formal description of SRNs and the numerical algorithms employed to solve the underlying Markov reward models may be found in [4].

3.1 Basic Terminology

A Petri net (PN) is a bipartite directed graph whose nodes are divided into two disjoint sets called *places* and *transitions*. Directed arcs in the graph connect places to transitions (called input arcs) and transitions to places (called output arcs). A *cardinality* may be associated with these arcs. A *marked Petri net* is obtained by associating *tokens* with places. A *marking* of a PN is the distribution of tokens in the places of the PN. In a graphical representation of a PN, places are represented by circles, transitions are represented by bars and the tokens are represented by dots or integers in the places. Input places of a transition are the set of places which are connected to the transition through input arcs. Similarly, output places of a transition are those places to which output arcs are drawn from the transition.

A transition is considered *enabled* in the current marking if the number of tokens in each input place is at least equal to the cardinality of the input arc from that place. The *firing* of a transition is an atomic action in which one or more tokens are removed from each input place of the transition and one or more tokens are added to each output place of the transition, possibly resulting in a new marking of the PN. Upon firing the transition, the number of tokens deposited in each of its output places is equal to the cardinality of

10

the output arc. Each distinct marking of the PN constitutes a separate state of the PN. A marking is reachable from another marking if there is a sequence of transition firings starting from the original marking which results in the new marking. The reachability set (graph) of a PN is the set (graph) of markings that are reachable from the other markings (connected by the arcs labeled by the transitions whose firing causes the corresponding change of marking). In any marking of the PN, multiple transitions may be simultaneously enabled.

Another type of arc in a Petri net is the *inhibitor* arc. An inhibitor arc drawn from a place to a transition means that the transition cannot fire if the place contains at least as many tokens as the cardinality of the inhibitor arc.

Extensions to PN have been considered by associating *firing times* with the transitions. By requiring exponentially distributed firing times, we obtain the *stochastic Petri nets*. The underlying reachability graph of a SPN is isomorphic to a continuous time Markov chain (CTMC). Further generalization of SPNs has been introduced in [1] allowing transitions to have either zero firing times (immediate transitions) or exponentially distributed firing times (timed transitions) giving rise to the generalized stochastic Petri net (GSPN). In this paper, timed transitions are represented by hollow rectangles while immediate transitions are represented by thin bars. The markings of a GSPN are classified into two types. A marking is *vanishing* if any immediate transition is enabled in the marking. A marking is *tangible* if only timed transitions or no transitions are enabled in the marking. Conflicts among immediate transitions in a vanishing marking are resolved using a *random switch* [1].

Although GSPNs provide a useful high-level language for evaluating large systems, representation of the intricate behavior of such systems often leads to large and complex structure of the GSPN. To alleviate some of these problems, several structural extensions to Petri nets are described in [5] which increase the modeling power of GSPNs. These include guards (enabling functions), general marking dependency, variable cardinality arcs and priorities. Some of these structural constructs are also used in stochastic activity networks (SANs) [16] and GSPNs [3]. Stochastic extensions were also added to GSPNs to permit the specification of reward rates at the net level, resulting in stochastic reward nets (SRN). All these extensions are described in the following subsections.

3.2   Marking dependency

Perhaps the most important characteristic of SRNs is the ability to allow extensive marking dependency. Parameters such as the rate of a timed transition, the cardinality of an input arc, or the reward rate in a marking, can be specified as a function of the number of tokens in some (possibly all) places. Marking dependency can lead to more compact models of complex systems.

3.2.1   Variable cardinality arc

In the standard PN and in most SPN definitions, the cardinality of an arc is a constant integer value [14]. If the cardinality of the input arc from place p to transition t is k, k tokens must be in p before t can be enabled and, when t fires, k tokens are removed from p. Often, *all* the tokens in p must be moved to some other place q. This behavior can be easily described in SRNs by specifying the cardinalities of the input arc from p to t and of

the output arc from t to q as $\#(p)$, the number of tokens in p. This representation is more natural, no additional transitions or places are required, and the execution time (to generate the reachability graph) is likely to be shorter.

The use of variable cardinality is somewhat similar to the conditional case construct of SANs [16]. We allow variable cardinality input, output arcs, and inhibitor arcs.

When the cardinality of the arc is zero, the arc is considered absent. The user of SRNs must be aware of the difference between defining the cardinality of an input arc as "$\max\{1, \#(p)\}$" or as "$\#(p)$". The former definition disables t when p is empty, the latter does not; the correct behavior depends on the particular application.

3.2.2   Priorities

Often, an activity must have precedence over another when they both require the same resource. Inhibitor arcs may be used to represent such constraints, but they may clutter the model. It is more convenient to incorporate transition priorities directly into the formalism. Traditionally, priorities have been defined by assigning an integer priority level to each transition, and adding the constraint that a transition may be enabled only if no higher priority transition is enabled. This can be generalized further by requiring only a partial order among transitions. Thus a priority relationship between two transitions t_1 and t_2 can be defined, for example as $t_1 > t_2$, implying that t_1 has higher priority compared to t_2. This added flexibility provides a simple way to model the situation where $t_1 > t_2$, $t_3 > t_4$, but t_1 has no priority relation with respect t_3 or t_4.

3.2.3   Guards

Each transition t may have an associated (boolean) guard $\mathcal{G}$. The function is evaluated in marking M when "there is a possibility that t is enabled", that is, when (1) no transition with priority higher than t is enabled in M; (2) the number of tokens in each of its input places is larger than or equal to the (variable) cardinality of the corresponding input arc; (3) the number of tokens in each of its inhibitor places is less than the (variable) cardinality of the corresponding inhibitor arc. Only then $\mathcal{G}(M)$ is evaluated; t is declared enabled in M iff $\mathcal{G}(M) = TRUE$. The default for $\mathcal{G}$ is the constant function $TRUE$.

The ability to express complex enabling/disabling conditions textually is invaluable. Without it, the designer might have to add extraneous arcs or even places and transitions to the SRN, to obtain the desired behavior. The logical conditions that can be expressed graphically using input and inhibitor arcs are limited by the following semantics: a logical "AND" for input arcs (all the input conditions must be satisfied), a logical "OR" for inhibitor arcs (any inhibitor condition is sufficient to disable the transition). For instance, a guard such as $(\#(p_1) \geq 3 \vee \#(p_2) \geq 2) \wedge (\#(p_3) = 5 \vee \#(p_4) \leq 1)$ is difficult to represent graphically.

3.3   Output measures

For a SRN, all the output measures are expressed in terms of the expected values of reward rate functions. Depending on the quantity of interest, an appropriate reward rate is defined.

Suppose X represents the random variable corresponding to the steady-state reward rate describing a measure of interest. A general expression for the expected reward rate in steady-state is

$$E[X] = \sum_{k \in \mathcal{T}} r_k \pi_k,$$

where $\mathcal{T}$ is the set of tangible markings (no time is spent in the vanishing markings), π_k is the steady-state probability of (tangible) marking k, and r_k is the reward rate in marking k.

Let $X(t)$ represent the random variable corresponding to the instantaneous reward rate of interest. The expression for the expected instantaneous reward rate at time t, becomes:

$$E[X(t)] = \sum_{k \in \mathcal{T}} r_k \pi_k(t),$$

where $\pi_k(t)$ is the probability of being in marking k at time t.

Let $Y(t)$ represent the random variable corresponding to the accumulated reward in the interval $[0, t)$ and Y represent the corresponding random variable for the accumulated reward until absorption. The expressions for the expected accumulated reward in the interval $[0, t)$ and the expected accumulated reward until absorption are:

$$E[Y(t)] = \sum_{k \in \mathcal{T}} r_k \int_0^t \pi_k(x) \, d\,x,$$

and

$$E[Y] = \sum_{k \in \mathcal{T}} r_k \int_0^\infty \pi_k(x) \, d\,x,$$

respectively.

The computation of the quantities π_k, $\pi_k(t)$, $\int_0^t \pi_k(x) \, d\,x$ and $\int_0^\infty \pi_k(x) \, d\,x$ using numerical techniques is described in [4].

4   An embedded system

In this section, we use SRNs to model the reliability of an embedded system.

4.1   System description

Consider a system consisting of an input processor, I, connected to three sensors S_1, S_2, and S_3, an output processor O, connected to an actuator with a spare, A_1 and A_2, a main processor, M, and a bus B (Figure 9).

The input processor polls its three sensors and takes three readings, performs some computation on them, such as taking the average or eliminating an outlier, then passes the result to the main processor. The main processor, in return, elaborates these results into commands to be passed to the output processor, which controls an actuator.

In a chemical plant environment, the sensor could be reading fluid level, temperature, or pressure, while the actuator could be controlling a valve. In an avionics system, the sensors could be reading speed, position, or direction, and the actuator could be controlling the flaps.
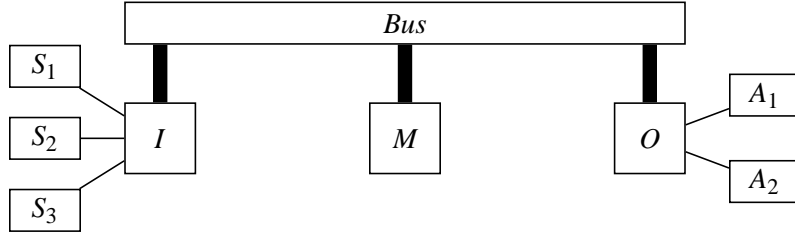
Figure 9: An embedded system.

During the normal operation of the system, the main processor uses a countdown timer to control the frequency at which readings are taken from the input processor and commands are issued to the output processor. The timer interval is set to τ at the beginning of every cycle.

The reliability of the system is affected by two types of faults. Processors, sensors, and actuators can fail, with rate λ_p, λ_s, and λ_a, respectively (the probability of bus failure is considered negligible, hence the bus is not explicitly modeled). The failure of a processor is always fatal. The sensors are used in triple modular redundancy (TMR), using the input processor as the voter, hence the system can survive the failure of one, but not two, sensors. The system is also able to survive the failure of one actuator, since a spare is available. The second type of fault is an intermittent, or transient, fault which might be experienced by the processors. If an input or output processor experiences a transient fault, it will reboot. During this time, the processor is unavailable, and, if the timer associated with the main processor expires, the system skips one monitoring cycle. A counter is used to keep track of how many consecutive cycles have been skipped. If this number exceeds a certain threshold *MaxCount*, the main processor decides that at least one of the other two processors has been unavailable for an excessive amount of time, and shuts the system down. We ignore the possibility of intermittent faults for the main processor, since, if the fault is indeed intermittent, the timer, which is associated with the main processor, can simply stop until the reboot is complete. In other words, the main processor must guess the status of the input and output processors, but not its own. The processor transient failure rate is δ_f, and the reboot rate is δ_r.

4.2 SRN model

The SRN in Figure 10 models the system just described. The meaning of each place and transition is shown in Table 1. Places *DownP*, *DownS*, and *DownA* are needed only because we want to distinguish between the cause of system failure (see Section 4.4). If we were only interested in the probability of the system being up, we could eliminate these places, thus merging all the absorbing (dead) markings into one, and reducing the size of the underlying CTMC.

Let's consider how transition *Timer* correctly updates the number of tokens in places *StartCycle*, *EndCycle*, and *Count*. First of all, *Timer* is always enabled, unless the system is down. When *Timer* fires, it removes the token which is either in place *StartCycle* or
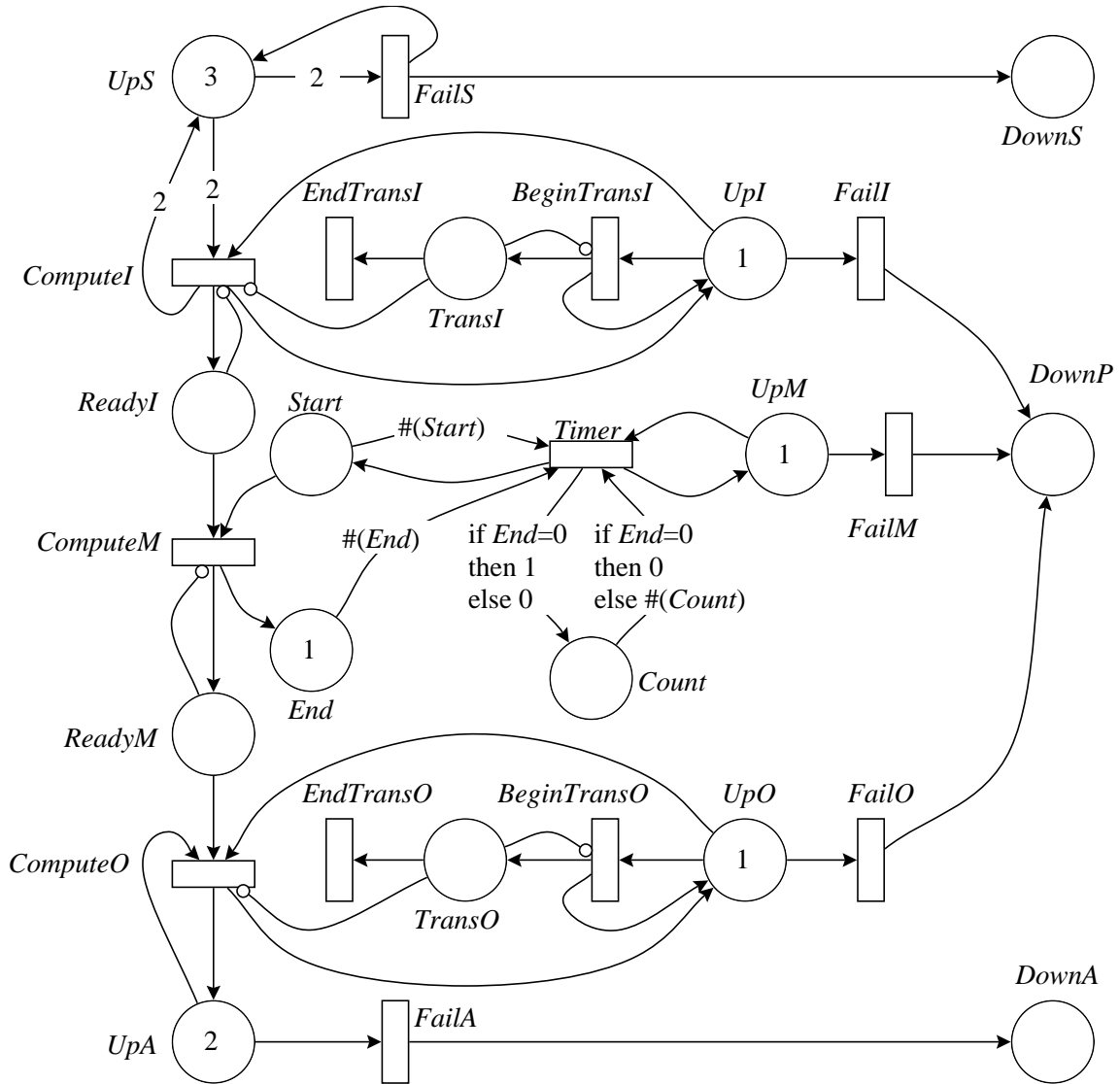
14

Figure 10: The SRN for the embedded system.

Place	Meaning
$UpX, X = I, M, O, S, A$	Number of working $I/M/O$ processors, sensors, actuators.
$DownX, X = P, S, A$	Number of down processors, sensors, actuators.
$TransX, X = I, O$	I/O processor is experiencing a transient fault.
$ReadyX, X = I, M$	I/M processor is ready to forward its results.
$Count$	Number of timeouts since the last successful cycle.
$Start$	A token here signifies that a cycle has begun.
End	A token here signifies the end of a successful cycle.
Transition	**Meaning**
$FailX, X = I, M, O, S, A$	Failure of a $I/M/O$ processor, sensor, or actuator.
$ComputeX, X = I, M, O$	Computation performed by a $I/M/O$ processor.
$BeginTransX, X = I, O$	Occurrence of a transient fault for a I/O processor.
$EndTransX, X = I, O$	Completion of the reboot process for a I/O processor.
$Timer$	Each firing corresponds to a cycle of the M processor.

Table 1: Meaning of the places and transitions for the SRN of the embedded system.

in place *EndCycle*, and puts it back in *StartCycle*. Also, the cardinality of the input and output arcs between *Count* and *Timer* is zero and one, respectively, if *EndCycle* is empty. This ensures that *Count* is incremented if the main processor has been unable to perform its computation by the time the timer elapses. If *EndCycle* contains a token, however, the cardinality of the input and output arcs between *Count* and *Timer* is the number of tokens in *Count* and zero, respectively, thus ensuring that *Count* is emptied (reset) if the main processor has successfully completed a cycle.

Also, note how the inhibitor arcs from *ReadyI* to *ComputeI* and from *ReadyM* to *ComputeM* ensure that at most one token can accumulate in them. This is essential to the correct representation of the model, since it ensures that the number of tokens in these two places is either zero or one.

On the other hand, the inhibitor arcs from *TransientI* to *ComputeI* and *TransientO* to *ComputeO* simply model the fact that, while a processor is rebooting, it cannot perform useful computation.

The system is up when the following boolean condition holds:

$$
\begin{aligned}
b = \quad & \#(UpS) \geq 2 \ \wedge \ \#(UpI) = 1 \ \wedge \ \#(UpM) = 1 \\
& \wedge \ \#(UpO) = 1 \ \wedge \ \#(UpA) \geq 1 \ \wedge \ \#(Count) \leq MaxCount
\end{aligned}
$$

We can then associate b as the guard for each transition in the SRN, or, in SPNP, simply define its negation $\bar{b}$ as a *halting condition* for the SRN.

4.3 Parametrization

The model is numerically very stiff, since it represents both failure events, normally of the order of months or years, and computation events, normally of the duration of seconds or even less. Since we are focussing on the reliability of this system, we can assume that the firing time for transitions *ComputeI*, *ComputeM*, and *ComputeO* is small compared to the

Parameter	Value	Meaning
$MaxCount$	2	system is down if there are three consecutive timeouts
λ_p	$1/31536000$ sec^{-1}	MTTF for a processor is one year
λ_s	$1/2592000$ sec^{-1}	MTTF for a sensor is one month
λ_a	$1/5184000$ sec^{-1}	MTTF for an actuator is two months
τ	$1/60$ sec^{-1}	timeout is one minute
δ_f	$1/86400$ sec^{-1}	one transient fault per day per processor, on average
δ_r	$1/30$ sec^{-1}	reboot time requires half a minute

Table 2: Parameters for the embedded system.

other activities, hence we can model these three activities with immediate transitions. This modification has two advantages: it results in a smaller tangible state space, and it reduces stiffness for the numerical solution. The approximation introduced is negligible, unless the timeout value τ is similar to the time required to perform the computation, but this would not be reasonable, since the timeout should be set so that it might elapse only when one of the processors is rebooting, not during fault-free operation.

The following parameters must then be assigned a value before the SRN can be evaluated:

- $MaxCount$, the maximum number of timeouts the main processor is willing to accept before deciding that at least one of the other two processors is faulty.

- λ_p, λ_s, and λ_a, the failure rates for the processors, sensors, and actuators, respectively.

- τ, the timeout interval for the main processor.

- δ_f and δ_r the transient failure and reboot rates, respectively.

We assume the values given in Table 2.

4.4   Output measures

We are interested in studying the probability that the system has failed by time t, for one of these possible causes:

- $\alpha_c = \Pr\{\#(Count) > MaxCount\}$: the number of consecutive timeouts has exceeded $MaxCount$.

- $\alpha_p = \Pr\{\#(DownP) = 1\}$: one of the processors has failed.

- $\alpha_s = \Pr\{\#(DownS) = 2\}$: two sensors have failed.

- $\alpha_a = \Pr\{\#(DownA) = 2\}$: both the actuator and the spare have failed.

To compute these probabilities, we can simply define four sets of 0/1 reward rates. For example, the reward rate for α_c is

$$r_i = \begin{cases} 1 & \text{if } \#(Count) > MaxCount \text{ in marking } i \\ 0 & \text{otherwise} \end{cases}$$

17

Figure 11 shows the probability of a failure due to the four cases, α_c, α_p, α_s, and α_a, computed using SPNP for $t = 0, 1, \ldots 24$ hours and $t = 0, 1, \ldots 30$ days. The reliability is given by one minus the sum of these four probabilities. Due to the characteristics of the TMR behavior, and to their low reliability (MTTF is one month), the sensors are the most critical components: they account for about 2/3 of the failures. On the other hand, the effect of timeouts is comparable to that of processor failures α_c and α_p are within a few percent of each other, when $MaxCount = 2$. If the risk of allowing the system to withstand a longer sequence of timeouts without shutting down is acceptable, $MaxCount$ could be increased. For example, by doubling its value ($MaxCount = 4$), α_c becomes negligible compared to the other causes of failure (see Figure 12).

5 Conclusions

In this paper we described the use of stochastic reward nets for the study of reliability models. Several interesting features of SRNs including marking dependency, guards, variable cardinality arcs and transition priorities were introduced. We also illustrated the specification of the output measures for the SRN models through the definition of reward rates.

References

[1] M. Ajmone Marsan, G. Conte, and G. Balbo. A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems. *ACM Trans. Comput. Syst.*, 2(2):93–122, May 1984.

[2] M. Ajmone Marsan, S. Donatelli, and F. Neri. GSPN models of multiserver multiqueue systems. In *Proc. Int. Conf. on Petri Nets and Performance Models*, pages 19–28, Kyoto, Japan, Dec. 1989. IEEE Computer Society Press.

[3] G. Chiola. A software package for the analysis of generalized stochastic Petri net models. In *Proc. Int. Workshop on Timed Petri Nets*, pages 136–143, Los Alamitos, CA, July 1985. IEEE Computer Society Press.

[4] 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*, volume 48. Springer-Verlag, Heidelberg, Germany, 1992.

[5] G. Ciardo, J. Muppala, and K. Trivedi. SPNP: Stochastic Petri net package. In *Proc. Int. Workshop on Petri Nets and Performance Models*, pages 142–150, Los Alamitos, CA, Dec. 1989. IEEE Computer Society Press.

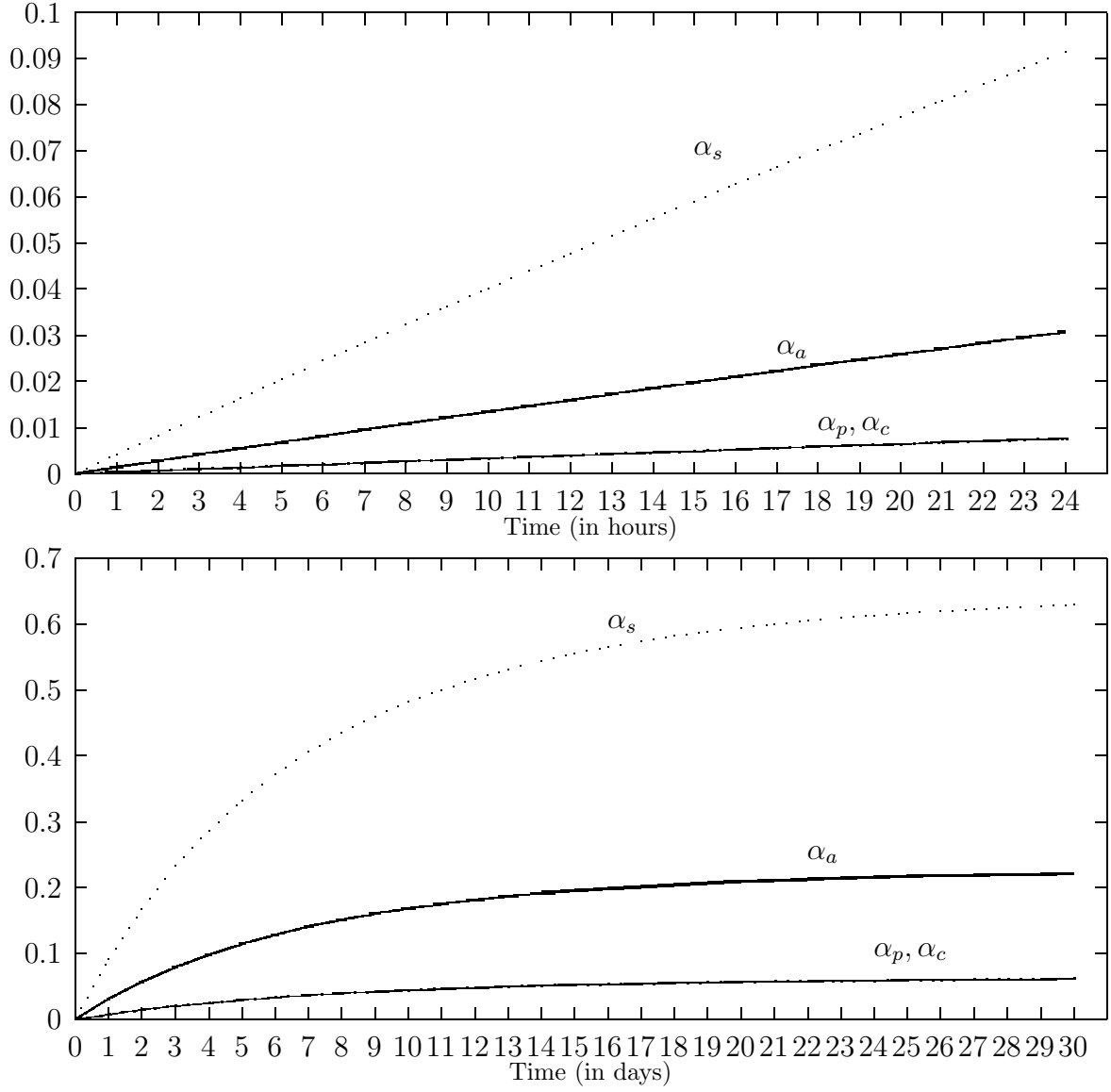[6] C. Colburn. *The Combinatorics of Network Reliability*. Oxford University Press, New York, NY, 1987.

Figure 11: Failure probabilities α_c, α_p, α_s, and α_a during the first day and month of operation.
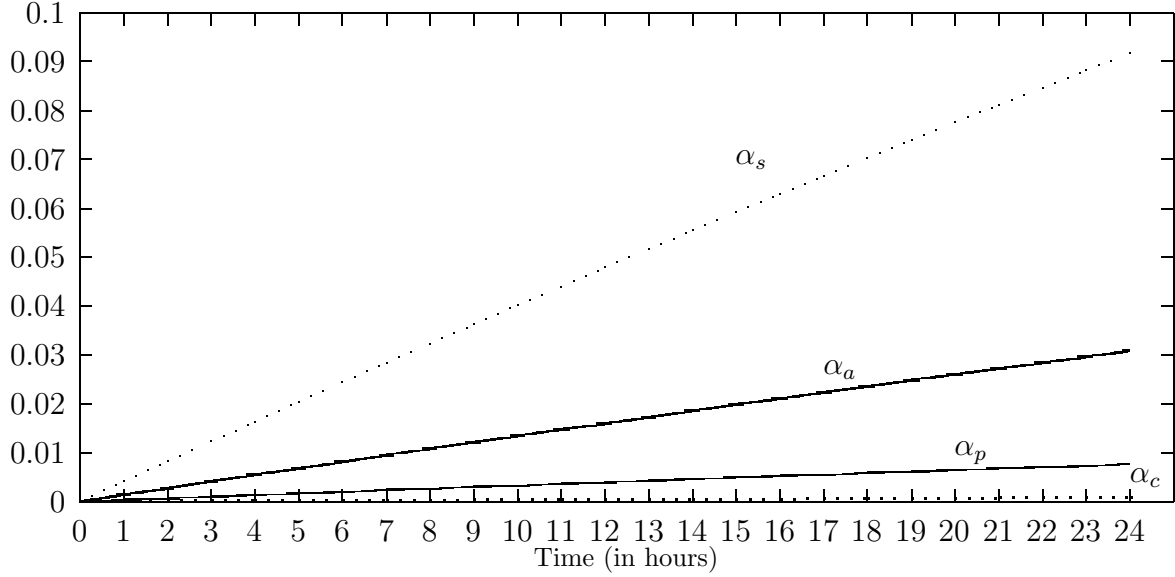
Figure 12: Failure probabilities α_c, α_p, α_s, and α_a during the first day of operation when $MaxCount = 4$.

[7] J. A. Couvillion, R. Freire, R. Johnson, W. D. Obal II, M. A. Qureshi, M. Rai, W. H. Sanders, and J. E. Tvedt. Performability modeling with ultrasan. *IEEE Software*, 8(5):69–80, Sep. 1991.

[8] J. B. Dugan, K. S. Trivedi, M. K. Smotherman, and R. M. Geist. The Hybrid Automated Reliability Predictor. *AIAA J. Guidance, Control and Dynamics*, 9(3):319–331, May-June 1986.

[9] A. Goyal, W. C. Carter, E. de Souza e Silva, S. S. Lavenberg, and K. S. Trivedi. The system availability estimator. In *Proc. Sixteenth Int. Symp. on Fault-Tolerant Computing*, pages 84–89, Los Alamitos, CA, July 1986. IEEE Computer Society Press.

[10] R. A. Howard. *Dynamic Probabilistic Systems, Vol II: Semi-Markov and Decision Processes*. John Wiley and Sons, New York, 1971.

[11] O. C. Ibe, K. S. Trivedi, A. Sathaye, and R. C. Howe. Stochastic Petri net modeling of VAXcluster system availability. In *Proc. Int. Workshop on Petri Nets and Performance Models*, pages 112–121, Los Alamitos, CA, Dec. 1989. IEEE Computer Society Press.

[12] S. W. Leu, E. B. Fernandez, and T. Khoshgoftaar. Fault-tolerant software reliability modeling using petri nets. *Int. J. Microelectronics and Reliability*, 31(4):645–667, 1991.

[13] J. F. Meyer. Performability: A retrospective and some pointers to the future. *Perf. Eval.*, 14(3-4):139–156, 1992.

[14] J. L. Peterson. *Petri Net Theory and the Modeling of Systems*. Prentice-Hall, Englewood Cliffs, NJ, USA, 1981.

[15] R. A. Sahner and K. S. Trivedi. Reliability modeling using SHARPE. *IEEE Trans. Reliability*, R-36(2):186–193, June 1987.

[16] W. H. Sanders and J. F. Meyer. METASAN: A performability evaluation tool based on stochastic activity networks. In *Proc. of the ACM-IEEE Comp. Soc. Fall Joint Comput. Conf.*, pages 807–816, Los Alamitos, CA, 1986. IEEE Computer Society Press.

[17] M. L. Shooman. *Probabilistic Reliability: An Engineering Approach.* McGraw-Hill, New York, NY, 1968.

[18] K. S. Trivedi. *Probability & Statistics with Reliability, Queueing, and Computer Science Applications.* Prentice-Hall, Englewood Cliffs, NJ, USA, 1982.

[19] K. S. Trivedi, J. K. Muppala, S. P. Woolet, and B. R. Haverkort. Composite performance and dependability analysis. *Perf. Eval.*, 14(3-4):197–215, 1992.

[20] M. Veeraraghavan and K. S. Trivedi. Hierarchical modeling for reliability and performance measures. In S. K. Tewksbury, B. W. Dickson, and S. C. Schwartz, editors, *Concurrent Computations: Algorithms, Architecture and Technology.* Plenum Press, New York, 1987.