STATE-SPACE SUPPORT FOR PATH-BASED REWARD VARIABLES[1]

W. Douglas Obal II[2]

Enterprise Storage Solutions Division
Hewlett-Packard Company
8000 Foothills Blvd. MS 5668
Roseville, California 95747-5668, USA

William H. Sanders

Center for Reliable and High-Performance Computing
Department of Electrical and Computer Engineering and
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
1308 W. Main St., Urbana, Illinois 61801, USA

Abstract

Many sophisticated formalisms exist for specifying complex system behaviors, but methods for specifying performance and dependability variables have remained quite primitive. To cope with this problem, modelers often must augment system models with extra state information and event types to support particular variables. This often leads to models that are non-intuitive, and must be changed to support different variables. To address this problem, we extend the array of performance measures that may be derived from a given system model, by developing new performance measure specification and model construction techniques. Specifically, we introduce a class of path-based reward variables, and show how various performance measures may be specified using these variables. Path-based reward variables extend the previous work with reward structures to allow rewards to be accumulated based on sequences of states and transitions. To maintain the relevant history, we introduce the concept of a path automaton, whose state transitions are based on the system model state and transitions. Furthermore, we present a new procedure for constructing state spaces and the associated transition rate matrices that support path-based reward variables. Our new procedure takes advantage of the path automaton to allow a single system model to be used as the basis of multiple performance measures that would otherwise require separate models or a single more complicated model.

1 Introduction

Many sophisticated formalisms now exist for specifying complex system behaviors, and many tools exist that can convert a model specified in the formalism to an underlying stochastic process that can be solved. Specification methods for performance and dependability variables, on the other hand, have remained quite primitive by comparison. For example, most stochastic Petri net (SPN) tools require a user to specify performance and dependability variables in terms of a rate defined on the states of the model, and possibly, an impulse defined on each event (e.g., transition in a SPN). The trouble with this approach is that the model, as naturally defined, may not support the desired measure. To overcome this limitation, one often has to add extra components to a model (e.g. places and transitions if modeling using SPNs) to collect the desired information. These components are not part of the system being modeled, and must change whenever one desires new information from the model.

It is thus often the case that several different models of a system must be built in order to obtain the desired performance measures. Changing the model can be a time-consuming procedure, since it then must be validated to guarantee that it is still an accurate representation of the system under study. We address this problem by extending performance measure specification and state-space construction procedures to allow more flexible use of a given model. This is made possible by 1) extending current reward variable specification methods to include variables that have "state" and can capture behavior related to sequences of events and states, and 2) extending current state-space construction algorithms that build stochastic processes that are tailored to the variable(s) of interest.

The use of performance measures to direct state-space construction is not new, but has been limited to supporting lumping based on symmetries for analytic models. In particular, [16] uses standard rate and impulse-based reward variables to put limits on the lumping that can be achieved because of symmetries in a model, and to support impulses that depend on particular activity completions. Path-based reward variables for impulses have been considered, but only to the extent that their use did not change the state space that is generated. Specifically, [13] considers the use of such variables, but limits their use

[1] This material is based upon work supported by DARPA/ITO under Contract No. DABT63-96-C-0069. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA/ITO.

[2] This work was done while the first author was a Ph.D. student in the Electrical and Computer Engineering Department of the University of Arizona, and a Visiting Scholar in the Coordinated Science Laboratory of the University of Illinois.

to impulses on sequences of instantaneous events in order not to change the set of (stable) states that is generated.

Our work extends previous work in two important ways. First, we provide support for a more general class of reward variables for a given system model. In particular, we support the definition of measures that depend on sequences of states and events that may occur. Examples of variables whose specification is facilitated by these methods include computations of probabilities of occurrence of particular recovery actions, which have multiple steps, and computation of measures related to consecutive cell loss in ATM networks, among others. We do this by introducing the "path automaton," a finite automaton that can be used to define rewards on sequences or sets of sequences of system model states and/or transitions (both timed and untimed). By building the required memory into the performance measure specification, we simultaneously accomplish two goals: we make more flexible the specification of complex performance and dependability variables, and we avoid the need to develop multiple system models. This approach offers the advantage of a single, smaller, system model that is easier to construct and validate. Multiple performance measures defined on multiple path automata may then be defined relative to the single system model.

Second, we provide procedures for automatic construction of a state space that supports the specified variables from the definition of the system model, path automata, and reward structures. Note that the choice of variables as well as system model determine the state space that is generated, and different variables result in different size state spaces for the same system model. In addition, these state generation procedures include automatic support for state-space truncation for the case of performance measures defined over intervals terminated upon satisfaction of a condition on the system model, such as entrance to a particular state or the occurrence of a sequence of states and/or transitions. This is made possible by the use of path automaton "final states," which are interpreted by the state-space construction procedure as indicating that the model state reached upon entry to the final state should not be explored any further.

The remainder of the paper is organized as follows. Section 3 introduces the new concept of a path-based reward variable, and Section 4 shows how various performance measures may be specified using path-based reward variables. Then, in Section 5, we present new procedures for automatically generating a state space that supports multiple path-based reward variables, and summarize the relevant numerical solution techniques. Section 6 gives an example model and some results on the variation of state space size for different performance measures.

2 Model Specification

In this section we review a simple model description formalism, first defined in [10].

Definition 1 *A* model *is a five-tuple* $(S, E, \varepsilon, \lambda, \tau)$ *where*

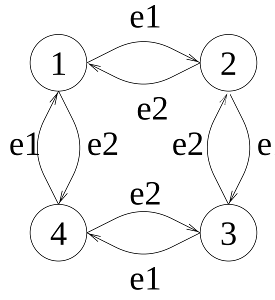
- S *is a set of state variables* $\{s_1, s_2, \ldots, s_n\}$ *that take values in* $I\!N$*, the set of nonnegative integers. The state of the model is defined as a mapping* $\mu : S \to I\!N$*, where for all* $s \in S$*,* $\mu(s)$ *is the value of state variable* s*. Let* $M = \{\mu \mid \mu : S \to I\!N\}$ *be the set of all such mappings.*
- E *is the set of events that may occur.*
- $\varepsilon : E \times M \to \{0, 1\}$ *is the event enabling function. For each* $e \in E$ *and* $\mu \in M$*,* $\varepsilon(e, \mu) = 1$ *if event* e *may occur when the current state of the model is* μ*, and zero otherwise.*
- $\lambda : E \times M \to (0, \infty)$ *is the transition rate function. For each event* e *and state* μ *such that* $\varepsilon(e, \mu) = 1$*, event* e *occurs with rate* $\lambda(e, \mu)$ *while in state* μ*.*
- $\tau : E \times M \to M$ *is the state transition function. For each* $e \in E$ *and* $\mu \in M$*,* $\tau(e, \mu) = \mu'$*, the new state of the model that is reached when* e *occurs in* μ*.*

Our objective in introducing Definition 1 is to simplify the presentation of our ideas for variable specification and state-space construction, which are independent of the details of the actual modeling language. While this formalism is well suited to our purpose, it is a low-level formalism, and leads to verbose and sometimes unwieldy model definitions. However, many different high-level modeling languages can be mapped to this formalism. We demonstrate one such mapping through our use of a stochastic activity network in Section 6.

We now present a simple example to illustrate this modeling formalism. We will also use the same example in Section 5 to illustrate our state-space construction methods. Consider the state transition diagram in Figure 1. This system has a single state variable and two events, each of which is enabled in each state. Thus $S = \{s_1\}$ and $E = \{e_1, e_2\}$. The state variable takes values in the set $\{1, 2, 3, 4\}$, leading to state mappings $\mu_1 = (s_1, 1)$, $\mu_2 = (s_1, 2)$, $\mu_3 = (s_1, 3)$ and $\mu_4 = (s_1, 4)$. Since each event is enabled in each state, $\varepsilon(e, \mu) = 1$ for all $(e, \mu) \in E \times M$. We define the rate of occurrence of e_1 in any of the four states to be ϕ, and the rate of e_2 to be θ. The definition of the model is given in the tables of Figure 1.

A model executes as follows. Starting in a state, μ, the enabled events, $E(\mu) = \{e \mid \varepsilon(e, \mu) = 1\}$, compete to cause the next state transition. Each event occurs in state μ at rate $\lambda(e, \mu)$, which means that the time until the event occurs, given that no other event occurs first, is exponentially distributed with density $\lambda(e, \mu)e^{-\lambda(e,\mu)t}$.

State transition diagram



Transition function and rate

State	Event	Next State	Rate
1	e_1	2	ϕ
	e_2	4	θ
2	e_1	3	ϕ
	e_2	1	θ
3	e_1	4	ϕ
	e_2	2	θ
4	e_1	1	ϕ
	e_2	3	θ

State variables

Identifier	Description
s_1	model state

Event set

Identifier	Description	Enabling Condition
e_1	clockwise	always enabled
e_2	counter clockwise	always enabled

Fig. 1. Example model

If event e occurs in μ, the next state is given by $\tau(e, \mu)$. The probability distribution over the set of states that may be reached in one transition from μ is determined in the usual way from the relative magnitudes of the event occurrence rates. In the example model of Figure 1, the mean holding time in each state is $\frac{1}{\phi+\theta}$ and the probability of a clockwise move via e_1 is $\frac{\phi}{\phi+\theta}$.

Models are created to answer questions about a system. We give such questions the generic name "performance measures," which we use in a broad sense, encompassing all the standard performance, dependability, and performability measures that have been defined in the literature. As pointed out by Meyer in [6], directly formulating performance measures defined at the system level in terms of a stochastic process representation of a model is not always feasible, due to the complexity of the mapping. Therefore, it is necessary (as well as desirable) to define performance measures at a high level. The concept of a "reward structure" has been established as a useful technique for specifying performance measures on models [1,5,14]. Defined at the model level, a reward structure associates reward accumulation rates with model states and impulse rewards with model events. Performance measures are then defined in terms of a reward structure and a variable specification, which together define a random variable called a *reward variable.* Common examples of such reward variables are the instantaneous reward at time t and the reward accumulated over the interval $[t, t+l]$.

3 Path-Based Reward Variables

We wish to evaluate a performance measure that is based on a sequence of model states and events. We call such a sequence a "path," and such measures "path-based performance measures." Formally,

Definition 2 *A* path, *$(\mu_1, e_1)(\mu_2, e_2), \ldots, (\mu_n, e_n)$, is a sequence of ordered pairs where μ_i is a model state and e_i is a model event.*

We say a path is *initialized* when the model enters the first state in the path. A path *completes* when the last pair in the sequence is satisfied. A path is *aborted* if the sequence is violated, for example if e_3 occurs in μ_2 instead of e_2. Definition 2 identifies a particular path, and there are many such paths in any given model. Sometimes the level of detail supported in this definition of path is not needed. For example, suppose that we are only interested in a sequence of events e_1, e_2, e_3, without regard to which states are visited. If there are many possible states that can be visited during this sequence of events, then many different paths must be specified. For situations like this, we need a convenient formalism for describing sets of paths.

To facilitate path-set specification, we introduce the "path automaton." The inputs to the automaton are a model event and the model state in which it occurred. For example, (e, μ), indicates that event e occurred in model state μ. The automaton state transition function defines the next state in terms of the current state and the input pair. Formally,

Definition 3 *A* path automaton *defined on a model, $(S, E, \varepsilon, \lambda, \tau)$, is a four-tuple, (Σ, F, X, δ), where*

- *Σ is the nonempty set of internal states;*
- *F is a (possibly empty) set of final states;*
- *$X = E \times M$ is the set of inputs; and*
- *$\delta : \Sigma \times X \rightarrow \Sigma \cup F$ is the state transition function, where for any internal state $\sigma \in \Sigma$ and input pair $x \in X$, $\delta(\sigma, x)$ identifies the next state.*

The path automaton executes as follows. Starting from an initial state $\sigma_0 \in \Sigma$, input pairs from the model are read, one for each state transition of the model. For each input pair $x \in X$, the state transition function $\delta(\sigma, x)$ identifies the next automaton state $\sigma' \in \Sigma \cup F$. If $\sigma' \in F$ then the path automaton halts. We say a path is *distinguished* by an automaton if completion of the path leaves the automaton in a final state.

We propose to evaluate path-based performance measures using path automata, together with "path-based reward structures."

Definition 4 *A* path-based reward structure *defined on path automaton* (Σ, F, X, δ) *is a pair of functions*

- $\mathcal{C} : \Sigma \times X \rightarrow I\!R$, *the* impulse reward function, *where for all internal states* $\sigma \in \Sigma$ *and input pairs* $x \in X$, $\mathcal{C}(\sigma, x)$ *is the impulse reward earned when the path automaton is in state* σ *and receives input pair* x.
- $\mathcal{R} : \Sigma \times M \rightarrow I\!R$, *the* rate reward function, *where for all internal states* $\sigma \in \Sigma$ *and model states* $\mu \in M$, $\mathcal{R}(\sigma, \mu)$ *is the rate at which reward is earned when the path automaton is in state* σ *and the model is in state* μ.

The path-based reward structure is a generalization of the standard (state-based) reward structure, since depending on the internal state of the path automaton, different rate rewards can be assigned to the same model state and different impulse rewards to the same model event. This is not possible with standard reward structures. On the other hand, any standard reward structure is easily represented using a path-based reward structure where the path automaton has only one state.

Reward variables can be constructed from the path-based reward structure and several random variables defined on the evolution of the model and the path automaton. The following random variables serve as components from which we may construct a broad array of performance measures:

- $I^t_{(\sigma,e,\mu)}$ is an indicator random variable representing the event in which at time t, the path automaton is in state σ and the last input pair was (e, μ);
- $J^{[t,t+l]}_{(\sigma,e,\mu)}$ is a random variable representing the total time during the interval $[t, t + l]$ that the automaton is in state σ and the last input pair was (e, μ); and
- $N^{[t,t+l]}_{(\sigma,e,\mu)}$ is an indicator random variable representing the number of times within the interval $[t, t + l]$ that the automaton is in state σ and receives input pair (e, μ).

Following the approach in [16], we can now define the reward variables we need for evaluating the various performance measures.

$$V_t = \sum_{(\sigma,x)\in\Sigma\times X} (\mathcal{C}(\sigma, x) + \mathcal{R}(\sigma, \mu)) \cdot I^t_{(\sigma,x)},$$

where μ is the state component in the pair $x = (e, \mu)$, is the instantaneous reward at time t. Note that in the usual case $\mathcal{C}$ would be zero for this type of variable. Otherwise the interpretation is that the impulse associated with the most recent event that occurred prior to t is accumulated along with the rate reward corresponding to the current automaton and model state pair.

$$Y_{[t,t+l]} = \sum_{(\sigma,x)\in\Sigma\times X} \mathcal{C}(\sigma, x) \cdot N^{[t,t+l]}_{(\sigma,x)} + \mathcal{R}(\sigma, \mu) \cdot J^{[t,t+l]}_{(\sigma,e,\mu)},$$

where e and μ are the components of x, is the reward accumulated over the interval $[t, t + l]$.

$$W_{[t,t+l]} = \frac{Y_{[t,t+l]}}{l}$$

is the time-averaged accumulated reward over the interval $[t, t + l]$.

So far we have considered reward variables that give us access to the value of the reward structure at fixed times or over intervals defined by fixed times. Some path-based performance measures, such as time to completion, call for an interval that is based on the random time at which some event occurs. In the literature on stochastic processes such random times are called *stopping times.* Performance measures based on stopping times are easily handled by our formalism for path-based reward variables. We define the random variable T_F to be the instant that the path automaton enters F, the set of final states. Now we can define V_{T_F} as the instantaneous reward immediately following entry to F, and $Y_{[t,T_F]}$ as the reward accumulated from time t until entry to F.

In the next section, we show how path-based reward variables can be used to obtain various performance measures.

4 Example Performance Measures

Given a model and a path, there are many different questions one might ask. First, we may want to know the probability of traversing the path. Given that the path is traversed, how long does it take? How many times was the path completed in some interval? What is the chance of finding the model in the middle of traversing the path, at some arbitrary time point? What is the total time spent traversing the path within some interval? In this section we show how all of these questions may be answered using path-based reward variables.

The model described in Figure 1 will be used to demonstrate the use of path-based reward variables. First we identify a path. Consider the following sequence of events: starting from state 1, the process visits states 2, 3, and 4 in

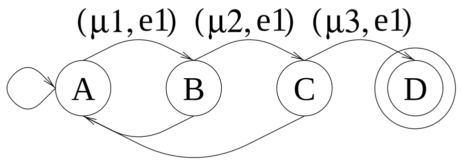


Fig. 2. Path automaton for probability of completion of $(\mu_1, e_1)(\mu_2, e_1)(\mu_3, e_1)$

exactly that order, with no other excursions. The path $(\mu_1, e_1)(\mu_2, e_1)(\mu_3, e_1)$ captures this sequence of events.

To compute the probability of traversing the path before time t, we use the path automaton shown in Figure 2, where a path completion causes a transition to a final state.

The state transition diagram in Figure 2 maps to the formal tuple-notation as follows. States of the automaton appear in the diagram as labeled circles. If there are final states, they are denoted by two concentric circles. Each arc between two states is labeled with the input that causes the transition represented by the arc. Unlabeled transitions are treated as "else" conditions. When we do not label an arc, we mean that the transition is taken if the input does not match a labeled arc.

The probability of traversing the path before time t is then the probability of finding the path automaton in its final state at time t. Thus we use the reward structure

$$\begin{aligned} \mathcal{C}(\sigma, x) &= 0 \\ \mathcal{R}(\sigma, \mu) &= \begin{cases} 1 \text{ if } \sigma = D \\ 0 \text{ otherwise} \end{cases} \end{aligned} \tag{1}$$

and evaluate $E[V_t]$, the expected value of V_t. Since $V_t = 1$ if the path completed and $V_t = 0$ otherwise, $E[V_t]$ is the probability that the path completes before t.

To compute the time to completion of the considered path, we use the path automaton in Figure 2, and the reward structure

$$\begin{aligned} \mathcal{C}(\sigma, x) &= 0 \\ \mathcal{R}(\sigma, \mu) &= \begin{cases} 1 \text{ if } \sigma \neq D \\ 0 \text{ otherwise.} \end{cases} \end{aligned}$$

The time to completion is the reward variable $Y_{[0,T_F]}$ where, for this automaton, $F = \{D\}$.

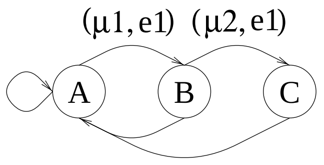


Fig. 3. Path automaton for number of completions of $(\mu_1, e_1)(\mu_2, e_1)(\mu_3, e_1)$

To count the number of completions of this path, we need to assign an impulse reward of 1 to (μ_3, e_1), but only if this event is preceded by $(\mu_1, e_1)(\mu_2, e_1)$. In addition, we do not want path completion to be a final state, since this performance measure is defined over multiple completions. The path automaton in Figure 3 is suitable for this performance measure. The reward structure for counting the number of completions of the path is

$$\begin{aligned} \mathcal{C}(\sigma, x) &= \begin{cases} 1 \text{ if } \sigma = C \text{ and } x = (\mu_3, e_1) \\ 0 \text{ otherwise} \end{cases} \\ \mathcal{R}(\sigma, \mu) &= 0. \end{aligned} \tag{2}$$

An impulse reward of 1 is assigned to the transition from path automaton state C to A if this transition is caused by event ϕ occurring in model state C. The number of completions of the path within the interval $[t, t+l]$ is $Y_{[t,t+l]}$.

The probability of finding the model on the path at time t and the time spent traversing the path within some interval are closely related; they have the same reward structure. Each performance measure can be computed by using the path automaton in Figure 3 and the reward structure

$$\begin{aligned} \mathcal{C}(\sigma, x) &= 0 \\ \mathcal{R}(\sigma, \mu) &= \begin{cases} 1 \text{ if } \sigma = A \text{ and } \mu = \mu_1 \\ 1 \text{ if } \sigma \in \{B, C\} \\ 0 \text{ otherwise.} \end{cases} \end{aligned}$$

The expected value of V_t gives us the desired result. To get the total time spent in states on the path in the interval $[t, t+l]$ we evaluate $Y_{[t,t+l]}$.

In most cases we want to be able to obtain as many different performance measures as possible from a single state space. Our method supports multiple performance measures on a single state space, but they must be "compatible," in a way which we will now describe precisely. A single state space can only support (one or more) fixed times, or one (random) stopping time. For this reason, if a stopping time, T_F, is used, we restrict the set of reward variables defined on a single state space such that each reward variable in the set is defined in terms of T_F. Thus two performance measures are *compatible* if they both refer to fixed times or they both refer to the same stopping time. Suppose that we define two path-based reward variables, v_1 and v_2, on a model, and that v_1 is defined on a path automaton that has a final state. For example, v_1 might measure the time to first completion of some path, p_1, while v_2 counts the number of completions of a different path, p_2. Clearly, v_1 and v_2 are most likely defined on different path automata. Yet, as stated previously, if a stopping time is present, all variables must be defined relative to that time in order to be supported by the same state space. Thus v_2 is subordinate to v_1, in the sense that v_2, if it is to be supported by the same state space as v_1, is understood to measure the number of completions of p_2 prior to the time of the first completion of p_1. For example, if we desire results for constant time t and stopping times T_{F_1} and T_{F_2}, three different state spaces are needed. Another ramification of this restriction is that a state space can only support one path automaton that has a nonempty set of final states.

This section has demonstrated how a variety of performance measures can be specified using path-based reward variables. Specific examples related to performance and dependability are given in Section 6. In the next section, we discuss the problem of constructing a state space that supports path-based reward variables.

5 State-Space Support

The first step in presenting the state-space construction method is to provide a precise definition of a state. We wish to allow multiple path-based reward variables to be associated with a given model. In general, this means that there will be multiple path automata and multiple reward structures to manage. Suppose that there are n different reward variables defined on a model. Each reward variable comprises a path automaton and a path-based reward structure. We index the path automaton and reward structure definitions by $i = 1, 2, \ldots, n$, so that, for example, δ_i is the state transition function for the i-th path automata. Thus we are led to the following definition of a state.

Definition 5 *For a model and a set of n path-based reward variables, a* state *is a four-tuple $(\sigma[], \mu, c[], r[])$ where*

- *$\sigma[]$ is an array of path automaton states, where $\sigma[i]$ is the internal state of the i-th path automaton;*
- *μ is the state of the model;*
- *$c[]$ is an array of impulse rewards for this state, where $c[i]$ is the impulse reward for the i-th reward variable; and*
- *$r[]$ is an array of rate rewards for this state, where $r[i]$ is the rate reward*

```
    $U$ : unexplored states
    $E(s)$ : events that may occur in $s$;
1.  Initial state $s_0 = (\sigma_0[], \mu_0, \mathbf{0}, r_0[])$
2.  $U = \{s_0\}$
3.  $S = \{s_0\}$
4.  while $U \neq \emptyset$
5.      choose $s \in U$
6.      $U = U - \{s\}$
7.      $E(s) = \{e \in E \mid \varepsilon(e, s.\mu) = 1\}$
8.      for each $e \in E(s)$
9.          $\mu' = \tau(e, s.\mu)$
10.         for $i = 1$ to $n$
11.             $\sigma'[i] = \delta_i(s.\sigma[i], s.\mu, e)$
12.             $c[i] = \mathcal{C}_i(s.\sigma[i], s.\mu, e)$
13.             $r[i] = \mathcal{R}_i(\sigma'[i], \mu')$
14.         $s' = (\sigma'[], \mu', c[], r[])$
15.         if $s' \notin S$
16.             $S = S \cup \{s'\}$
17.             if $\sigma'[1] \notin F$
18.                 $U = U \cup \{s'\}$
19.             add arc from $s$ to $s'$ with rate $\lambda(e, s.\mu)$
```

Fig. 4. Procedure for constructing a state space that supports multiple path-based reward variables

for the i-th reward variable.

Note that $r[]$, the array of rate rewards, is determined by σ and μ, the automaton and model states, so it does not add to the state space. However, it is convenient for our presentation to include the complete reward structure in the notion of state. In an implementation, some additional flexibility can be achieved by defining rate rewards separately from the states. As stated at the end of Section 4, the set of path automata supported by a single state space can only include one automaton where F is nonempty. We use the convention that if there is a path automaton with $F \neq \emptyset$, it takes the first position in the array of automaton definitions.

Figure 4 shows a procedure for constructing state spaces that support multiple path-based reward variables. In line 1, the initial state s_0 is identified by the initial state of each path automaton, the initial model state, and the initial reward structure values for each reward variable. By convention, the initial impulse rewards are zero, which is indicated by $\mathbf{0}$. In lines 2 and 3 we initialize the set of unexplored states and the set of visited states. At line 4, starting from the initial state, a breadth-first search of the state space is conducted.

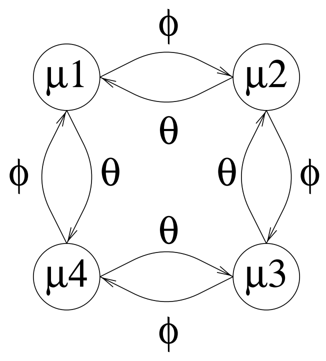


Fig. 5. Markov process state transition diagram for simple model

First, we choose a member, s, of the set of unexplored states (line 5). The next step (line 7) is to find the set of events, $E(s)$, that may occur in the current state. For each event the next model state, μ', is found (lines 8–9) by evaluating the model state transition function τ. At this point we have all the information needed to compute the reward structure for the new state. For each reward variable, we compute (lines 10–13) the next automaton state, the impulse reward for e occurring in μ, and the rate reward for the new automaton state, model state pair. These elements form the new state, s', constructed in line 14.

Then (line 15) we check to see if we already visited this state. If not, s' is a new state and we add it to the set of states (line 16). As long as the first path automaton in the array did not enter a final state as a result of the event that just occurred, we also add the new state to the set of unexplored states (line 18). We do not add the state to the unexplored set if it corresponds to a final automaton state, because in this case we want the state to be an absorbing state in this state space, even if it would not be an absorbing state in the model. We do this in order to support evaluation of reward variables defined in terms of stopping times. We will discuss this further at the end of this section. Finally, in line 19 we add a transition from the current state s to the new state s'. The process just described repeats until there are no remaining unexplored states.

To demonstrate the construction procedure, we use the simple example of Figure 1. The Markov process constructed directly from the model in Figure 1, without accounting for any performance measures, is shown in Figure 5. We now construct state spaces for two path-based performance measures, one of which is defined on an interval determined by a stopping time. The first example is the probability of path completion. To support this variable, we use the path automaton of Figure 2 with the reward structure in (1). The state space is shown in Figure 6. Note that the initial state is $(A, \mu_1, 0, 0)$.

As a second example, we consider the number of completions within an interval $[t, t+l]$. For this performance measure, we use the path automaton in Figure 3

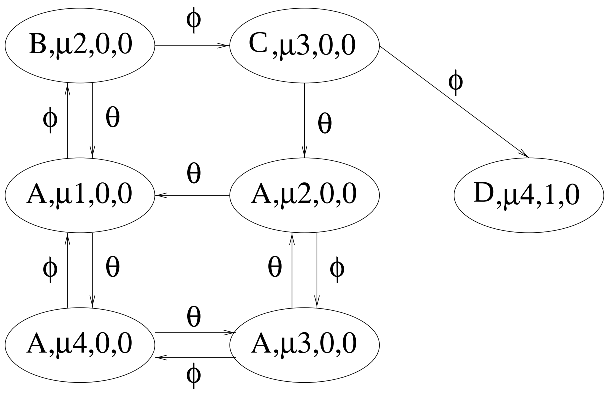


Fig. 6. State-space supporting probability of completion of path (μ_1, e_1), (μ_2, e_1), (μ_3, e_1) by time t

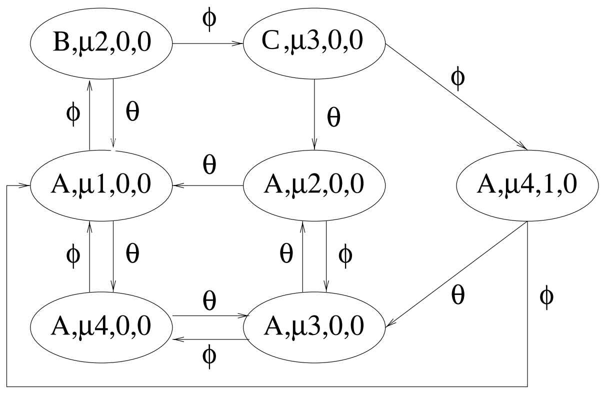


Fig. 7. State-space supporting number of completions of path (μ_1, e_1), (μ_2, e_1), (μ_3, e_1)

and the reward structure in (2). The constructed state space is shown in Figure 7. As in Figure 6, the initial state is $(A, \mu_1, 0, 0)$.

After the state space has been constructed, the model needs to be solved for the performance measures of interest. Table 1 summarizes examples of solution procedures that can be used to obtain the reward variables defined in Section 3. In the table, π is the vector of steady-state occupancy probabilities and SOR stands for Successive Over-Relaxation. The distribution of the instantaneous reward variable V_t is completely determined by the state occupancy probabilities at time t, which may be computed using uniformization for time t or by standard Gauss-Seidel or SOR for the limiting case $V_{t\to\infty}$. For $Y_{[t,t+l]}$, the expected value is easily obtained using uniformization. Techniques for calculating the distribution of $Y_{[t,t+l]}$ are much more sophisticated but available (see, for example, [3,4,8,11,12,17]).

To evaluate $Y_{[0,T_F]}$, the methods described in [2,5,9,18] can be used to compute the distribution of the reward accumulated until absorption. For $Y_{[t,T_F]}$ we need the state occupancy probabilities at time t, which we then use as the initial state distribution for the methods used for $Y_{[0,T_F]}$. The time-averaged

Table 1
Methods for evaluating the reward variables

Reward Variable	Solution Method	Obtainable Information
V_t	Uniformization	Distribution
$V_{t\to\infty}$	Gauss-Seidel, SOR	Distribution
$Y_{[t,t+l]}$	Uniformization	Expected value
	Special methods (e.g. [3,4,8,11,12,17])	Distribution
$Y_{[t,T_F]}$	Linear system (e.g. [2,5,9,18])	Moments
$Y_{[t\to\infty,t+l]}$	$Y_{[0,l]}$ starting with π	Distribution

accumulated reward, $W_{[t,t+l]}$, is easily derived from Y.

6 Fault-Tolerant Computing Example and Results

The size of the state space that is needed to support a path-based reward variable clearly depends on the underlying model and the nature of the path automaton. In this section we introduce a larger model and investigate the variation in the size of the state space required for several path-based reward variables.

As mentioned in Section 2, the modeling formalism introduced there and used to develop the variable specification and state-space construction procedures is not intended to be used for large models. For the example model in this section, we use a stochastic activity network (SAN) [7] to represent the system. Each place in the SAN is a state variable in the formalism of Section 1. The possible stable markings of the SAN correspond to state variable mappings in the formalism. We now define the mapping between events in our formalism and events in a SAN. The completion of a timed activity, a case selection, and a sequence of instantaneous activity completions, if any instantaneous activities are enabled, is mapped to an event in our formalism. Using SAN terminology, we map each possible “stable step” [15] to an event in the formalism. The state transition function is determined by the execution rules of the SAN, as are the event-enabling function and the event rate function.

We model a computer system designed to function in the presence of software faults. A stochastic activity network model of the system is shown in Figure 8. The system has two modes of operation: normal and diagnostic. In normal mode, there is one token in place *Normal*, and in diagnostic mode, there are zero tokens in *Normal*. Jobs arrive to be processed according to a Poisson process defined by timed activity *arrival*. The finite buffer capacity is modeled

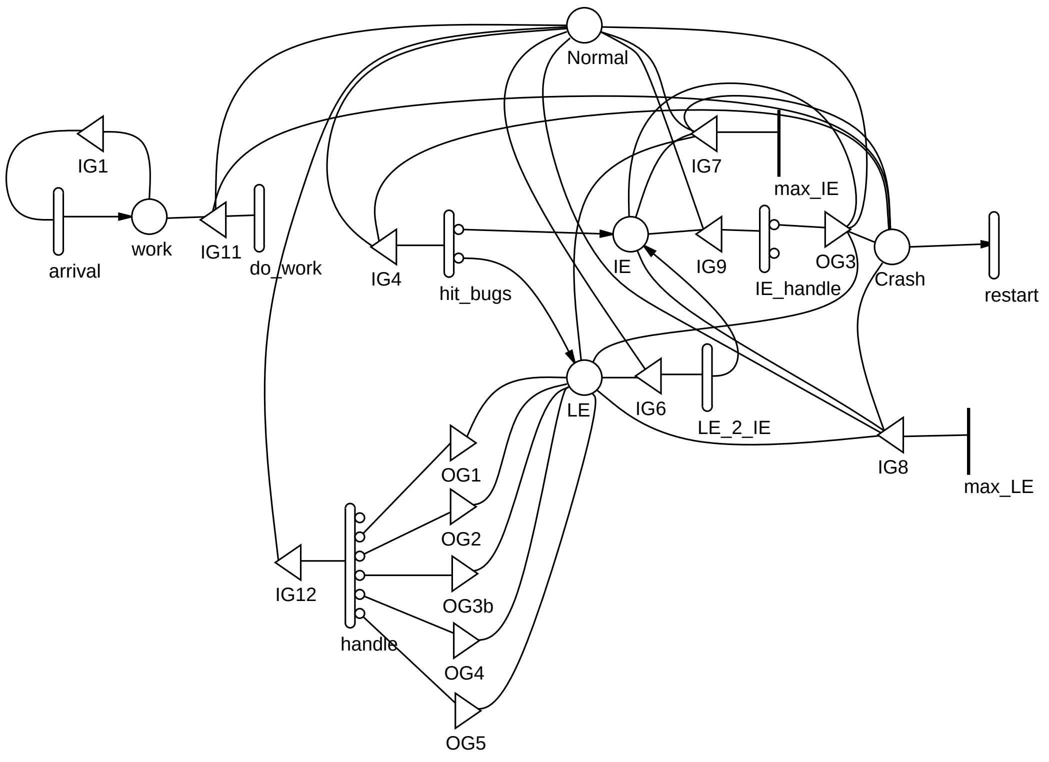


Fig. 8. SAN model of fault-tolerant computer system

by input gate *IG1*. When there are jobs to be processed, the system stays in normal mode. Jobs are processed at a rate defined by timed activity *do_work*. During operation in normal mode various program bugs may be encountered, at a rate governed by timed activity *hit_bugs*. The result of exercising a bug may be an immediately effective error (case 1), or a latent error. An example of an immediately effective error is an address violation; corruption of a data structure is an example of a latent error.

Immediately effective errors are handled by special error-handling routines that attempt to recover from the error. The error-handling operation is modeled by timed activity *IE_handle* and its two cases. If the error is mishandled (case 1), the system crashes. Furthermore, coincident errors can not be handled, so if an immediately effective error arrives before the last one is handled, the system crashes. This transition is modeled by instantaneous activity *max_IE* and input gate *IG7*. Latent errors do not generate any immediate problems, but the number of latent errors in the system (number of tokens in place *LE*) affects the rate at which immediately effective errors occur. This is modeled by the marking-dependent rate of timed activity *LE_2_IE*. We assume that the maximum number of latent errors that can actually be tolerated is five, so if the number of tokens in place *LE* exceeds five, input gate *IG8* enables instantaneous activity *max_LE* to model the resulting system crash.

Upon completing the available workload, the system runs a diagnostic program

Table 2
Model parameters for the fault-tolerant system

Parameter	Description	Value
IE_handle rate	Rate at which bug interrupts are handled	10
LE_2_IE rate	Rate at which LE manifest as IE	0.2
MAX_IE	Maximum number of coincident IE	1
MAX_LE	Maximum number of LE that can be tolerated	5
MAX_WORK	Maximum number of jobs in the system	60
PDR (used in *handle*)	Probability of detecting and removing a LE	0.95
PHE (case 1, *IE_handle*)	Probability of handling an IE	0.95
PIE (case 1, *hit_bugs*)	Probability of IE	0.05
handle rate	Rate at which diagnostic program executes	2
arrival rate	Rate of job arrivals	1
hit_bug rate	Rate at which bugs are encountered	0.1
do_work rate	Job processing rate	2
initial_workload	System starts with one job available	1
restart rate	Rate for system restart completion	0.05

to check for errors. Input gate *IG11* controls the switch to diagnostic mode. When the completion of timed activity *do_work* leaves zero tokens in place *work*, *IG11* removes the token from place *Normal*, thus initiating diagnostic mode. This program is capable of detecting and removing latent errors. The latent error detection and removal process is modeled by timed activity *handle* and its six cases. The outcome of the diagnostic program depends on the number of latent errors in the system. This dependency is modeled by the six marking-dependent case probabilities defined for timed activity *handle*.

Finally, when the system crashes, it automatically begins a restart operation, which is essentially a reboot. The time it takes to restart the system is modeled by timed activity *restart*. Upon restarting, the system is as good as new.

The model parameters we used are listed in Table 2.

There are many different performance measures that can be defined for this system. Standard non-path-based measures include the distribution of the number of jobs waiting to be serviced, the availability of the system, and the fraction of time spent running diagnostics.

Several path-based performance measures are also interesting. A path automa-

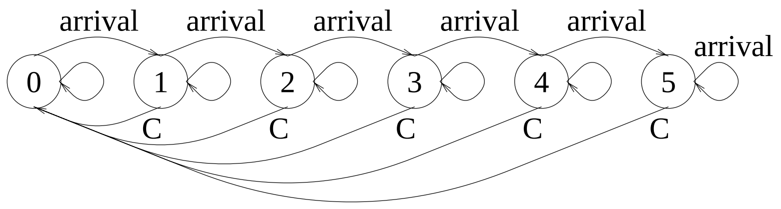


Fig. 9. Path automaton for detecting runs of length $k \geq 5$

ton is well-suited to the problem of evaluating the number of times the buffer blocks as a result of a given number of consecutive job arrivals before a job completion. We refer to a sequence of k consecutive job arrivals as a *run of length k*. Figure 9 shows the state transition diagram of a path automaton that can be used to count the number of runs of length $k \geq 5$. Starting from automaton state zero, the automaton records arrivals by increasing its state for each arrival. If a job completion occurs, the automaton resets. If any other event occurs, the automaton remains in its current state.

Using the path automaton of Figure 9, we define three reward variables. The first (3) is simply the number of times the system blocks, which translates to the number of times a job arrives and causes the buffer to be full.

$$\mathcal{C}(\sigma, x) = \begin{cases} 1 \ \mu(work) = W_{\max} - 1 \text{ and} \\ \quad e = \text{arrival} \\ 0 \text{ otherwise.} \end{cases} \tag{3}$$
$$\mathcal{R}(\sigma, \mu) = 0$$

The second (4) reward variable has an impulse of 1 on the event in which the automaton reaches state five, which corresponds to a run length of at least 5.

$$\mathcal{C}(\sigma, x) = \begin{cases} 1 \text{ if } \sigma = 4 \text{ and } e = \text{arrival} \\ 0 \text{ otherwise.} \end{cases} \tag{4}$$
$$\mathcal{R}(\sigma, \mu) = 0$$

Finally, the third variable (5) records an impulse of 1 each time the buffer becomes full following a run length of at least 5.

$$\mathcal{C}(\sigma, x) = \begin{cases} 1 \text{ if } \sigma = 4 \text{ or } \sigma = 5 \text{ and} \\ \quad \mu(work) = W_{\max} - 1 \text{ and} \\ \quad e = \text{arrival} \\ 0 \text{ otherwise.} \end{cases} \tag{5}$$

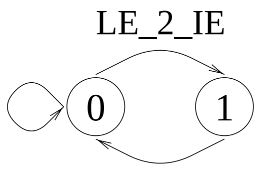


Fig. 10. Path automaton for number of crashes caused by a latent error becoming effective and being mishandled

$$\mathcal{R}(\sigma, \mu) = 0$$

Blocking events are primarily of interest for performance reasons. From a dependability standpoint, we are more interested in the processes that govern the transition from normal operation to a system crash. Understanding the dominant failure mechanism in a system is important in deciding where to spend time and money to improve the dependability. An example of a path-based dependability measure is the number of crashes in an interval that result from a latent error becoming effective and then being mishandled by the error-handling routines. This event is modeled by completion of timed activity *LE_2_IE* followed by completion of *IE_handle* and selection of case 1. The path automaton that captures this sequence of events is shown in Figure 10. The automaton spends most of its time in state 0, but when timed activity *LE_2_IE* completes, the automaton moves to state 1. Upon the next event, the automaton returns to state 0. The reward structure

$$\mathcal{C}(\sigma, x) = \begin{cases} 1 \text{ if } \sigma = 1 \text{ and} \\ \quad \text{e} = \textit{handle\_IE}, \text{ case one} \\ 0 \text{ otherwise.} \end{cases}$$
$$\mathcal{R}(\sigma, \mu) = 0$$

assigns an impulse reward of 1 to the event in which *LE_2_IE* completes and is immediately followed by *IE_handle*, case 1.

Tables 3 and 4 show the sizes of the state spaces required to support the example performance and dependability measures. To see how the state space grows with increasing path length, we constructed state spaces for extended run lengths. Figure 11 shows the state space growth corresponding to increasing run length. The fact that the curve in Figure 11 levels off may, at first glance, be surprising. The explanation for this behavior is that as the run length increases there are fewer model states that may be occupied for each state in the path automaton. For example, the model can not be on a run of length > 40 and have a queue length less than 40. Thus there are fewer additional states as the run length increases.

Table 3
State space sizes for example performance and dependability measures

Description	State Space Size
Number of model states	1586
Number of crashes via the sequence LE_2_IE, IE_handle, case 1	1952
Number of runs in the workload process that are ≥ 2 and Number of overflows preceded by a run ≥ 2	3826
Number of runs in the workload process that are ≥ 3 and Number of overflows preceded by a run ≥ 3	4567
Number of overflows and Number of overflows preceded by a run ≥ 4	4567
Number of runs in the workload process that are ≥ 4 and Number of overflows preceded by a run ≥ 4	5295

In Table 4, we have listed the results obtained for the expected number of blocking events in $[0, 500]$ that follow runs with length $\geq N$. The expected number of blocking events in $[0, 500]$ was also calculated from these state spaces, so they are larger (due to the additional impulses recorded by blocking events not caused by runs) than would have been required if we only wanted results for one variable. To validate our code, we obtained results for the SAN model using the *UltraSAN* modeling environment (see Appendix B). The results we obtained using *UltraSAN* agreed with those obtained using our code. However, we had to employ a different approach using *UltraSAN*. We added two extra places to the model and some extra code to input gates *IG1* and *IG11* in order to accumulate a count of buffer blocking events that followed runs $\geq N$. By specifying a reward structure that assigned a rate reward equal to the count of buffer blocking events, we could then solve for the expected instantaneous rate of reward at time 500 to get the expected number of buffer blocking events in $[0, 500]$. This is computed from the transient state

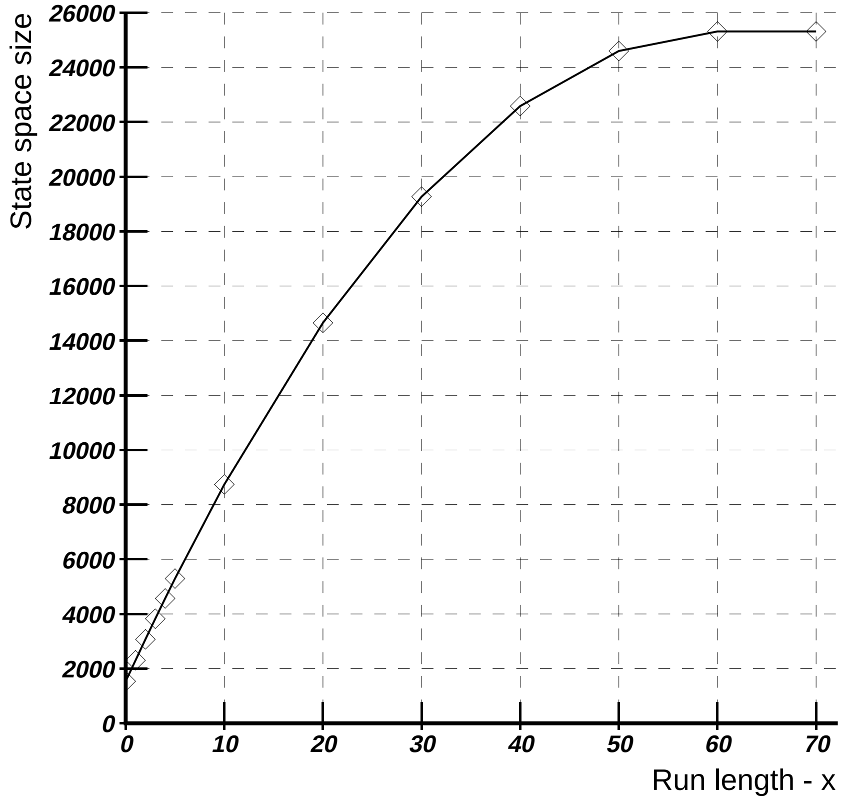


Fig. 11. State space size versus arrival run length ($\geq$), extended to the full range of the queue

occupancy probabilities. To obtain a finite state space using this approach, it was necessary to limit the possible number of blocking events. We could do that for this model, since the number of such events is so low (rarely > 1). However, for longer run lengths or in situations where the number of blocking events is expected to be large, this approach will fail because the state space will grow too large.

On the other hand, with path-based reward models, the problem is mitigated by the fact that we can employ an accumulated reward solver and count impulse rewards generated by the path-based reward structure. The state space still increases with the run length, but we do not have the additional growth that would have been caused by having to accumulate the number of blocking events. The expected value of accumulated reward over an interval of time is also computable using uniformization, and the computation times are similar.

Figure 12 shows the graph of the data in Table 4. The graph has an interesting shape, with steep slopes in the initial and final stages, but a relatively flat region in between. The shape of this curve may be understood by considering the distribution of the queue length. Figure 13 shows the probability mass at

Table 4
Expected number of blocking events in $[0, 500]$ following runs $\geq N$

N	States	Result
Base Model	1525	N/A
N=1	2305	0.302
N=2	3072	0.199
N=3	3826	0.164
N=4	4567	0.151
N=5	5295	0.147
N=10	8740	0.138
N=20	14655	0.119
N=30	19270	0.100
N=40	22585	0.083
N=50	24600	0.069
N=60	25315	0.010
N=70	25315	0.000

$N = 60$, which is large due to the relatively long restart time following a crash. Once the buffer is blocked, the probability of short runs leading to additional blocking events is relatively high, which (together with the fact that all long runs finish with short runs) accounts for the initial stage of the results curve shown in Figure 12. But as reflected in the results, the probability decreases sharply as soon as the queue begins to empty. In the middle portion of the run length range, the curve is relatively flat, which is explained by the fact that the probability of each queue length in this range is low. The small negative slope is due to the increasing run length. Finally, Figure 14 shows the concentration of probability mass at the lower queue lengths. The concentration of probability mass around lower queue lengths results in a low value for the expected queue length. We computed the steady-state expected queue length as 4.5, with a standard deviation of 10.7. The steep slope in the final stage of the curve in Figure 12 is due to the relatively large probability masses at the lower queue lengths. To understand this, note that the probability that the queue will be in a state that admits a run of length 55 is around 0.86, but the probability of finding the queue in a state that admits a run of length 59 is only 0.6. For each step below the expected queue length, a relatively large probability mass is lost, which leads to the steep slope in the final stage of the curve.

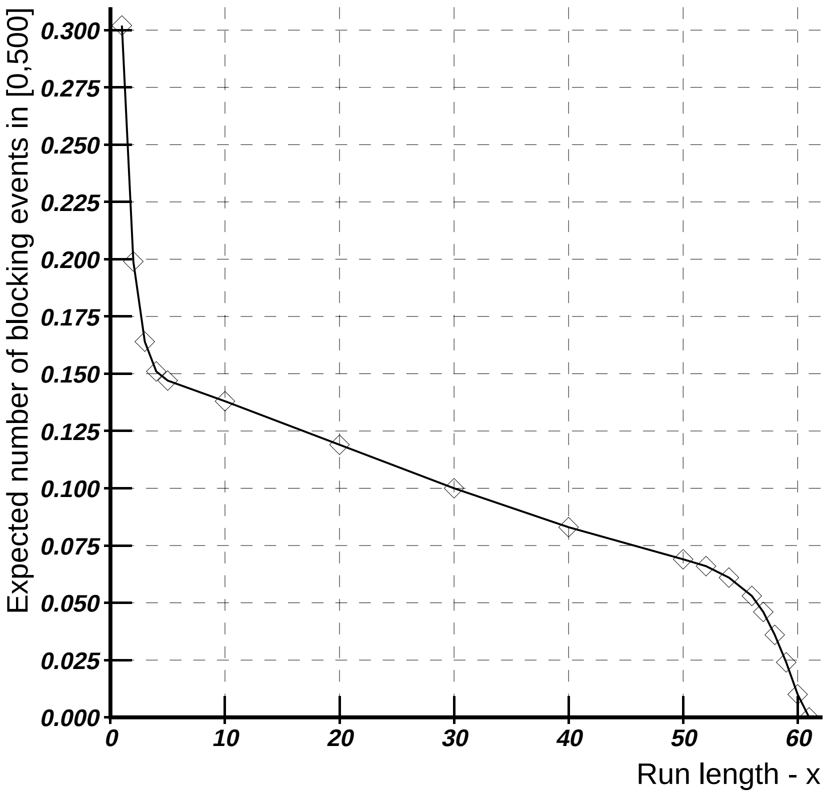


Fig. 12. Results for expected number of runs in [0,500] that exceed N

7 Conclusion

This paper presents a new performance/dependability measure specification technique and state space construction procedure that, together, solve the problem of supporting a broad array of performance measures from a given system model. Using this new approach, performance measures based on model states, model events, or sequences of (model state, model event) pairs can be supported by a single system model. Furthermore, the performance measures may be evaluated in steady state, at an instant of time, or over an interval of time defined by fixed endpoints or in terms of a (random) stopping time.

More specifically, we have shown how to specify path-based performance measures using the notion of a path automaton. With the current model state and model event as the basis for path automaton state transitions, we can use the path automaton to specify in a compact way measures that depend on particular sets of sequences of model states and events, as well as those that can be represented by standard reward variables. In addition, we have developed a state generation procedure that makes use of one or more path automata and the system model to generate a state space that is tailored to the variables of

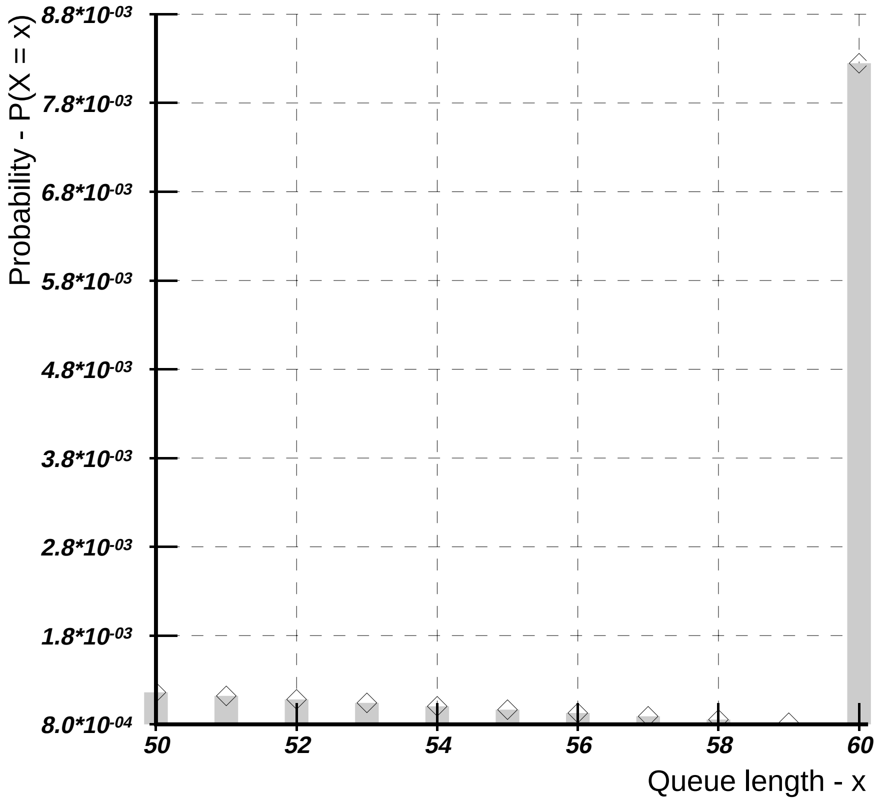


Fig. 13. The probability mass at $N = 60$ explains the higher values and steeper slope for $N \leq 5$

interest. In this way, a single system model can support many different performance/dependability variables, and drive the generation of many different state-level models, depending on the measure(s) of interest.

Finally, we have illustrated the use of these measure specification and state generation techniques on several small examples, to show the versatility of the approach, and on a more realistic fault-tolerant computing example, to show the variation in state-space size that can be expected for different measures. These results show the diversity of measures that can be supported by a single model.

Acknowledgments

We acknowledge Dan Deavours for giving us his SPN state space construction code, which served as a convenient starting point for our prototype implementation, and G. P. Kavanaugh for helpful discussions on the nature of path-based performance measures.

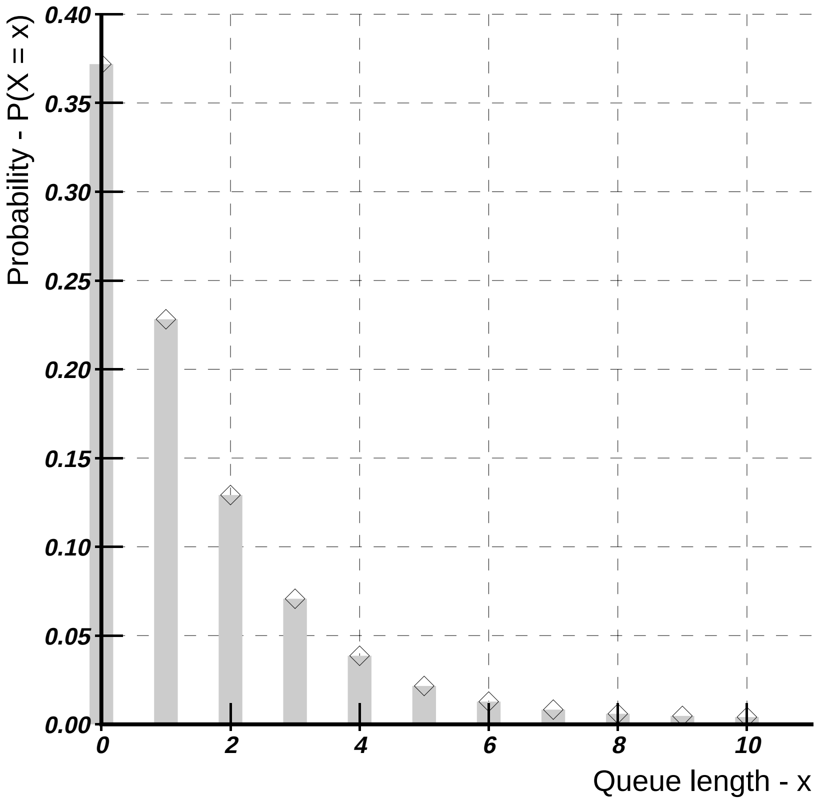


Fig. 14. The concentration of probability mass at the lower queue length values explains the roll-off that begins at $N \geq 55$. The queue length has a mean of 4.5 and a variance of 114.

References

[1] G. Ciardo, A. Blakemore, P. F. J. Chimento, J. K. Muppala, and K. S. Trivedi, “Automated generation and analysis of Markov reward models using stochastic reward nets,” in *IMA Volumes in Mathematics and its Applications*, Springer-Verlag, New York, 1992. vol. 48.

[2] E. Çinlar, *Introduction to Stochastic Processes*, Prentice-Hall, Englewood Cliffs, New Jersey, 1975.

[3] E. de Souza e Silva and R. Gail, “Calculating transient distributions of cumulative reward,” in *Proceedings of Sigmetrics/Performance-95*, pp. 231–240, Ottawa, Canada, May 1995.

[4] L. Donatiello and V. Grassi, “On evaluating the cumulative performance distribution of fault-tolerant computer systems,” *IEEE Transactions on Computers*, vol. 40, no. 11, pp. 1301–1307, November 1991.

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

[6] J. F. Meyer, "Probabilistic modeling," in *The Twenty-Fifth International Symposium on Fault-Tolerant Computing (FTCS-25)*, pp. 88–95, Pasadena, California, June 27–30, 1995.

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

[8] H. Nabli and B. Sericola, "Performability analysis: A new algorithm," *IEEE Transactions on Computers*, vol. 45, no. 4, pp. 491–494, April 1996.

[9] M. F. Neuts, *Matrix-Geometric Solutions in Stochastic Models: An Algorithmic Approach*, Johns Hopkins, Baltimore, Maryland, 1981.

[10] W. D. Obal II and W. H. Sanders, "Detecting and exploiting symmetry in discrete-state Markov models," Technical Report, 1997.

[11] K. R. Pattipati, Y. Li, and H. A. P. Blom, "A unified framework for the performability evaluation of fault-tolerant computer systems," *IEEE Transactions on Computers*, vol. 42, no. 3, pp. 312–326, March 1993.

[12] M. A. Qureshi and W. H. Sanders, "A new methodology for calculating distributions of reward accumulated during a finite interval," in *Proceedings of the 26th International Symposium on Fault-Tolerant Computing (FTCS-26)*, pp. 116–125, Sendai, Japan, June 25–27, 1996.

[13] M. A. Qureshi, W. H. Sanders, A. P. A. van Moorsel, and R. German, "Algorithms for the generation of state-level representations of stochastic activity networks with general reward structures," *IEEE Transactions on Software Engineering*, vol. 22, no. 9, pp. 603–614, September 1996.

[14] W. H. Sanders and J. F. Meyer, "A unified approach for specifying measures of performance, dependability, and performability," in *Dependable Computing for Critical Applications*, A. Avizienis and J. C. Laprie, editors, volume 4 of *Dependable Computing and Fault-Tolerant Systems*, pp. 215–238, Springer Verlag, Vienna, 1991.

[15] W. H. Sanders, *Construction and Solution of Performability Models Based on Stochastic Activity Networks*, PhD thesis, University of Michigan, 1988.

[16] W. H. Sanders and J. F. Meyer, "Reduced base model construction methods for stochastic activity networks," *IEEE Journal on Selected Areas in Communications, special issue on Computer-Aided Modeling Analysis, and Design of Communication Networks*, vol. 9, no. 1, pp. 25–36, January 1991.

[17] R. M. Smith, K. S. Trivedi, and A. V. Ramesh, "Performability analysis: Measures, an algorithm, and a case study," *IEEE Transactions on Computers*, vol. 37, no. 2, pp. 406–417, 1987.

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

Contact Information

William H. Sanders

Coordinated Science Laboratory and

Department of Electrical and Computer Engineering

University of Illinois at Urbana-Champaign

1308 W. Main St.

Urbana, Illinois 61810 USA

Telephone: +1 (217) 333-0345

FAX: +1 (217) 244-3359

`whs@crhc.uiuc.edu`