Model-Based Evaluation: From Dependability to Security

David M. Nicol, *Fellow, IEEE,* William H. Sanders, *Fellow, IEEE,* and Kishor S. Trivedi, *Fellow, IEEE*

***Abstract*— The development of techniques for quantitative, model-based evaluation of computer system dependability has a long and rich history. A wide array of model-based evaluation techniques are now available, ranging from combinatorial methods, which are useful for quick, rough-cut analyses, to state-based methods, such as Markov reward models, and detailed, discrete-event simulation. The use of quantitative techniques for security evaluation is much less common, and has typically taken the form of formal analysis of small parts of an overall design, or experimental *red team*-based approaches. Alone, neither of these approaches is fully satisfactory, and we argue that there is much to be gained through the development of a sound model-based methodology for quantifying the security one can expect from a particular design. In this work, we survey existing model-based techniques for evaluating system dependability, and summarize how they are now being extended to evaluate system security. We find that many techniques from dependability evaluation can be applied in the security domain, but that significant challenges remain, largely due to fundamental differences between the accidental nature of the faults commonly assumed in dependability evaluation, and the intentional, human nature of cyber attacks.**

Dependability Evaluation, Security Evaluation, Performability Evaluation, Stochastic Modeling

I. Introduction

Computer system and network security is an issue of increasing practical concern and research attention. As a research discipline, computer security is a venerable one, with its own culture, assumptions, and language. The increased emphasis on system security has brought new researchers from different backgrounds to the field, bringing different perspectives and different skillsets. All of this is for the good, since new viewpoints can lead to new insights.

Most of the older work in computer security focused on details in complex protocols or details in complex systems, for the simple reason that the root causes of security gaps are often found in the failures associated with such details. Later work expanded the attention to system-level security, that is, to the study of how systems can be designed to be secure, in the sense that they perform their intended function in spite of possible malicious attacks. New work is examining intrusion tolerance (e.g. [1], [2], [3], [4]), which is a means of designing systems to continue to perform their intended function, in spite of partially successful attacks.

No system-level methodology currently exists that can quantify the amount of security provided by a particular system-level approach. So far, most attempts at validation of security have been qualitative, focusing more on the process used to build a system that should be secure. Since it is impossible in practice to build a perfectly secure system, it is important to be able to quantitatively validate the efficacy of systems intended to be secure. Efforts aimed at quantitative validation of security have usually been based on formal methods (e.g., [5]), or have been informal, using "red teams" to try to compromise a system (e.g., [6]). Both approaches, while being valuable in identifying system vulnerabilities, have their limitations, especially when they are applied to large systems. Only recently, due to efforts led by researchers accustomed to quantifying other system measures, have attempts been made to quantify measures associated with system security.

This paper surveys concepts and methodologies for the evaluation of system dependability [7], [8], [9], and summarizes how these are now being extended to evaluate system security. These techniques differ from most other treatments of system security in that they frequently use stochastic modeling. Stochastic assumptions are needed to describe systems that have yet to be built, and for systems whose specific vulnerabilities remain unknown. In such cases it is appropriate to make stochastic assumptions about the introduction and discovery of vulnerabilities, about attacker behavior, about system behavior (in terms of the effects the exploited vulnerabilities have on it, and in terms of system's responses to attacks), and about transient periods of vulnerability, and to solve (or simulate) the model for stochastic measures. Stochastic models are also very useful for sensitivity analysis.

In this survey, we find that there is much in classical dependability analysis that can be transferred to security analysis. However, we also find that there are attributes of security that can't be integrated naturally into a dependability framework. In addition, we point out that the root causes of system failure in the context of classical dependability are fundamentally different from the root causes of security violations, in ways that impact the usefulness of the models we develop to describe those failures. We conclude the paper with ideas for future work in the area.

II. Measures of Dependability and Security

Engineers have long used models to evaluate system designs. The models employed typically focus on the questions that are most pressing to an engineer (e.g., Will the bridge collapse? Are structural reinforcements needed to meet stress tolerances? Will the computer system provide a specified response time? Is the flight control system able to tolerate 3 simultaneous equipment failures?). Today, we are faced with pressing questions about computer system and network *security*, which is defined informally as the resilience of a computer system or network to malicious attacks. Our thesis is that modeling can help one design and evaluate systems or networks that are intended to be secure, just as modeling has helped us provide and evaluate other system properties. Furthermore, we see interesting similarities between computer system failures due to intentional attacks and system failures due to accidental component failures; we thus see value in exploring how evaluation techniques developed to quantify system dependability might be extended to quantify system security.

However, we also see that security brings new issues to be considered, most notably related to causes of component and system failure. Dependability analysis to date usually assumes that failures are caused by random events in hardware or rare events in software, and that this randomness can be quantified (even if correlated) in a way that permits the determination of system-level properties. Security analysis must assume that failures are caused by human intent, resulting in security failures that are definitely correlated, that depend in subtle ways on system state, and that attackers learn over time. While such events might appear to be random, as perceived by an outside observer, they tend to depend on each other in subtle ways that make them difficult to represent accurately using classical stochastic models. To highlight these differences, we review classical dependability measures, pointing out challenges that arise when they are applied to systems that fail due to malicious attacks.

Reliability is the probability that a system performs a specified service throughout a specified interval of time. Reliability analysis therefore depends on stochastic models of the frequency, duration, and intensity of faults in hardware and software. While one can certainly *assume* some probabilistic structure when modeling cyber attacks, the problem of developing and validating good stochastic models is very much an open issue. However, it is one we must solve if we are to use classical reliability analysis to predict reliability in the face of security breaches. Having said that, it may be possible to use a conjectured probabilistic specification of the occurrence of cyber attacks to support sensitivity analysis of a system's reliability or availability. Indeed, several initial attempts have been made to quantify system security using ideas developed to quantify the effect of accidental failures (see, for example [10], [11], [12], [13], [14]).

Availability is a quantification of the alternation between proper and improper service, and is often expressed as the fraction of time that a system can be used for its intended purpose during a specified interval of time or in steady state. Challenges similar to those described above in the context of reliability analysis apply when evaluating a system's availability under malicious attack. Furthermore, as we consider the impact of security on availability, we see that a system's availability may be affected in several ways by a cyber attack; for example, it may be affected by the attack's own impact on the system and by the efforts to diagnose the attack and restore system service following the attack. Therefore, availability analysis of a system under malicious attack needs to specify explicitly how (and how long) a system remains unavailable following a successful attack.

Safety is the probability that a system does not fail in a manner that causes catastrophic damage during a specified period of time. Since system safety depends on the effect of a system failure rather than on the cause of the failure, one can easily imagine quantifying system safety in the context of cyber attack. While the safety of data from accidental erasure has certainly been a consideration in safety analysis of information systems, when we add security considerations, we will also need to consider the security of sensitive data in the event of a security breach as a component of safety. For example, the exposure of very sensitive data (e.g., private keys) might enable an attacker to cause catastrophic damage.

Performability [15] quantifies system performance in the presence of failures (either component or system). Performability analysis is often carried out by specifying a set of *structural* states for a system, each state corresponding to a configuration that results in a particular system performance, and specifying how the system changes state (often in the form of transition rates). Once these states and transitions have been identified, one then quantifies the amount of performance obtained in each state by specifying the rate at which *reward* is obtained in each state, and the amount (*impulse*) of reward that is obtained when a particular state transition is taken.

Using that mathematical structure, it is possible to specify performability measures in terms of the amount of reward accumulated during a specified interval of time (where the start or length of the interval can tend to infinity), or the rate of accumulation of reward at a specified instant of time or in steady state. (More details can be found in the next section.) Note that this method of specifying reward is general enough to support a wide variety of dependability and performability measures [16], [8], [17]. For example, if the rate at which reward is earned in an operational state is 1, while the reward rate earned in a non-operational state is 0, the expected reward accumulated over a specified interval of time is just the expected amount of time the system was available over that epoch; in other words, it is the system's expected interval availability. If, instead, we define the rate reward for each state as the rate at which work is accomplished in that state and define state in the model in a manner that captures changes in the rate at which work can be done that are due to component failures, then the reward accumulated during an interval specifies a measure of the quality of service provided to a user.

The ease with which measures can be specified in the framework described above is quite useful for the analysis of security breaches. For example, consider a denial-of-service cyber attack. The impact of that type of cyber attack and the system's attempts to cope with it can be reflected in the reward accumulated while spending time in states that reflect the attack. Another application of performability concepts in the security context follows from the fact that security measures may make a system harder to use. For example, security measures can slow the inherent rate at which work may be accomplished. Performability measures and analysis techniques thus provide a framework for considering the impact cyber attacks have on overall system performance. However, performability analysis faces the same principal challenge faced by reliability and availability analyses, namely the development of meaningful stochastic descriptions of events that occur during a cyber attack.

System security includes attributes in addition to those we have described above in the context of dependability. In particular, *data confidentiality* means that a system does not allow protected data to be read in an unauthorized fashion, while *data integrity* means that a system does not allow protected data to be modified in an unauthorized fashion. A breakdown in confidentiality or integrity need not imply a failure of reliability or availability, at least not in their usual senses. It may qualify as a safety issue, and could, depending on the structure of the associated measure, be expressed as a performability measure.

However, confidentiality and integrity are arguably different measures from those normally considered by dependability analysis, insofar as they are not concerned with system behavior so much as certain system properties. Likewise, *non-repudiation* is a system property that prevents future false denial of involvement by either party in a transaction. *Authentication* is a related property, by which the claimed identity of a party to a transaction can be independently verified. There are no obvious connections of authentication or non-repudiation to classical dependability measures.

Another important property in the security domain is survivability. As defined in [18], *survivability* is the capability of a system to fulfill its mission, in a timely manner, in the presence of attacks, failures, or accidents. Since survivability measures quantify the ability of a system to perform an intended function, modeling approaches that are applicable to availability and performability evaluation can be adapted for survivability evaluation. Recent work in quantifying survivability is found in [19], [20], [21], among other places.

Models for security analysis must describe how and when security breaches occur; they must describe the impact on the system when they do, as well as the mechanisms, effects, and costs of system recovery, system maintenance, and defenses. Stated as such, those are identical to the requirements of dependability models. However, there need to be significant differences in the nature and details of security models. This is most pointedly seen when we consider the introduction of security failures. A leading source of security vulnerability is misconfiguration. Failures due to misconfiguration can of course happen in other contexts, but a distinguishing feature in the security context is that some external agent must deliberately exercise the vulnerability in order for the failure to occur.

Latent software faults (e.g., buffer overflow problems) are another cause of security failure. Any given fault has specific idiosyncratic behaviors and requirements for accessing and exploiting it. Like misconfiguration, a security penetration made possible by a latent software fault does not occur accidently, but is actively induced by an attacker. Furthermore, a security penetration may require an attacker to exercise several vulnerabilities before compromising a prized asset (such as root access). This coupling of system vulnerabilities and attackers' exploitation of them distinguishes security failures from the types of failures traditionally considered by dependability analysis. The key issue is that of how to characterize attacker behavior.

III. Model Representation / Analysis Techniques

Research in dependability analysis has led to a variety of models, each focusing on particular levels of abstraction and/or system characteristics. As we extend that type of analysis into the security domain, we again find utility in diverse model types. We now review important classes of model representation and report on how they are being extended.

A. Combinatorial Methods

In contrast with state-space models, combinatorial models do not enumerate all possible system states to obtain a solution. Instead, simpler approaches are used to compute system dependability measures. Despite several extensions that have been made to combinatorial models, they do not easily capture certain features, such as stochastic dependence and imperfect fault coverage. We present a brief overview of combinatorial models.

1) Reliability Block Diagrams (RBD): An RBD is a graphical structure with two types of nodes: blocks representing system components and dummy nodes for connections between the components. Edges and dummy nodes model the operational dependency of a system on its components. At any instant of time, if there exists a path in the system from the start dummy node to the end dummy node, then the system is considered operational; otherwise, the system is considered failed. A failed component blocks all the paths on which it appears. RBDs thus map the operational dependency of a system on its components and not the actual physical structure of the system.

Series-Parallel RBDs are useful not only because they are very intuitive, but also because they can be solved in linear time [22]. Such RBDs are quite frequently used in reliability and availability modeling [8], [9], and many software packages exist that support construction and solution of RBD models (e.g., [23], [22]). We have yet to see an application of RBDs in security modeling. For such an application to be possible, one would need to create a compositional theory of security. At a glance, it seems that such a theory ought to have different semantics; in particular, an architecture needs to isolate an insecure component, not just provide a replicate in parallel.

2) Fault Trees (FTs): A fault tree is an acyclic graph with internal nodes that are logic gates (e.g., AND, OR, k-of-n) and external nodes (leaves or basic events) that represent system components. The edges represent the flow of failure information in terms of Boolean entities (TRUE and FALSE or 0s and 1s). Typically, if a component has failed, a TRUE is transmitted; otherwise, a FALSE is transmitted. The edge connections determine the operational dependency of the system on the components. At any instant of time, the logic value at the root node determines whether or not the system is operational. If shared (repeated) nodes (nodes that share a common input) are not allowed, then the acyclic structure is a rooted tree.

Fault trees without shared nodes are equivalent to series-parallel RBDs [9], but when shared nodes (or repeated events) are allowed, fault trees are more powerful [24]. Many solution algorithms exist for fault trees with repeated events, including those based on sums of disjoint products (see [25]) and binary decision diagrams (e.g., [26], [27]). Fault trees have been extensively used in reliability and availability modeling (e.g., [28], [29], [30], [31]), safety modeling (see [32]), and modeling of software fault tolerance (e.g., [33]).

Fault trees have been extended to include various types of gates, such as priority AND gates, sequence dependency gates, exclusive OR gates, and inhibitor gates [34]. There have been extensions to include imperfect coverage [35], multistate systems [36], and phased mission systems [37]. A large number of software packages that support the construction and solution of fault trees are available (e.g., [38], [39], [22]). The main difficulty of using combinatorial methods in practice is the common assumption that all basic events must be statistically independent.

3) Attack Trees: Attack trees are closely related to fault trees, in that they consider a security breach as a system failure, and describe sets of events that can lead to system failure in a combinatorial way. An attack tree thus models all possible attacks against a system, just as a fault tree models all failures. They provide a formal, methodical way to describe the security of systems and subsystems based on various types of attacks (originally described in [40]) using graphics that are somewhat different than those that have become standard for fault trees.) In an attack tree, the attacks to a system are represented in a tree structure, with the goal as the root node and the different ways to achieve that goal as leaf nodes. The security of a large system can be modeled with a set of attack trees, where the root of each tree represents an attack that can significantly damage the system's operation.

Structures and Semantics: In an attack tree, each non-leaf node represents an attack goal (or subgoal), and leaf nodes are atomic attacks. There are two kinds of non-leaf nodes: AND nodes and OR nodes. An AND node represents an attack goal for which a set of subgoals must be achieved in order for the attack to succeed. These attack subgoals are represented by the AND node's children. An OR node represents an attack goal that can be achieved in several ways, which are represented by the OR node's children.

Attack trees can be represented graphically or textually. A representation of an AND node is shown in Figure 1. The

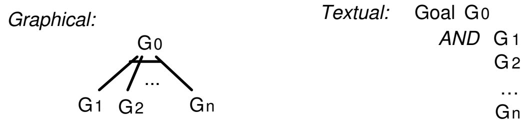

Fig. 1. AND node

figure shows a goal G_0 that can be achieved if the attacker achieves each of G_1 through G_n.

A representation of an OR node is shown in Figure 2. It shows a goal G_0 that can be achieved if the attacker achieves any one of G_1 through G_n.

Assigning Node Values: Once the attack tree has been created, different values can be assigned to the leaf nodes. These values can be:

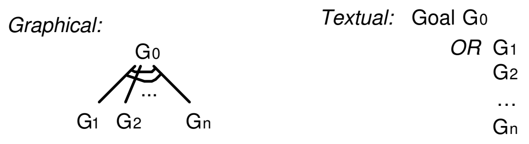

Fig. 2. OR node

1) *Boolean (e.g., possible vs. impossible):* A possible node represents a feasible attack scenario; an impossible value means the attack cannot be carried out in the current situation. Some other Boolean values include easy vs. not easy, expensive vs. not expensive, intrusive vs. non-intrusive, legal vs. illegal, and special equipment required vs. no special equipment required.
2) *Continuous:* Sometimes it is not enough to use Boolean values to describe the attacks. For example, we may want to know the probability that an attack goal can be achieved. We can do so by assigning continuous values to each leaf node. These values may include the cost in dollars to attack/defend, the effort spent to achieve/repulse, the probability that the attack will succeed/fail, and the likelihood that the attacker will try the attack.

Evaluating Attack Trees: After assigning values to each leaf node, it is possible to propagate the node value up to the root of the tree. A node's value is a function of its children's values. Depending on whether a node is an AND node or an OR node and the nature of the assigned values, the calculation rules may differ. For example, if the possible/impossible values are under consideration, the AND node's value is the Boolean and of all values of its children, while the OR node's value is the Boolean or of all values of its children. When cost value is considered, the value of the AND node is the sum of the values of its children, and the value of the OR node is the minimum of the values of its children.

The attack tree can be used to evaluate different aspects of the system security, depending on the kind of value that is assigned to the leaf nodes. If a possible/impossible value is assigned, one can enumerate all sets of possible atomic attacks that achieve the attack goal; if a probability value is assigned, one can use an attack tree to evaluate the probability that the attack goal can be achieved. If a cost value is assigned, an attack tree can be used to evaluate the minimum cost needed to reach an attack goal.

Since atomic attacks can have multiple attributes, each leaf node can have several different value types. Therefore, an attack tree can be used to combine these values and help users learn more about a system's vulnerabilities. For example, if one assigns a possible/impossible value as well as a cost-in-dollars value to each node, one can use the attack tree to find the lowest-possible-cost attack sets for the system; if a probability value as well as a special equipment value is assigned, one can obtain the most-probable attack sets with no special equipment required.

Although fault trees are used primarily in dependability analysis and attack trees are used primarily in the security context, they share much in common. They have the same tree structures; they both contain AND nodes and OR nodes, which are the two most commonly used node types in both trees; and they have similar calculation rules to propagate node values up to the root. Therefore, many techniques used in fault tree analysis can be applied to analyze attack trees. For example, the SDP, BDD, or factoring methods used to compute system reliability can also be used in attack trees to obtain the probability that the attack goal will be reached; the minimum cut-set and minimum path sets analysis from fault trees can be used to find all sets of atomic attacks that achieve the goal; and, similar concepts and computations of importance measures in fault tree analysis can be applied to attack trees to evaluate the impact of certain atomic attacks on the overall system security.

Attack trees thus provide a systematic way to describe the security vulnerabilities, thus making it possible to assess risks and make security decisions. They capture knowledge and expertise in a reusable form; once the attack tree for a certain security feature has been built, it can be included as part of a larger attack tree for a system that uses the security feature.

B. Model Checking

Another important type of dependability and security analysis is based on a reachability analysis of the model state space, an activity sometimes known as *model checking*. The general approach has a long history in hardware verification. The idea is to analyze the state space implied by some formal expression of the system. Certain states reflect deleterious conditions; model-checking algorithms explore the entire state space, report states of interest as they are uncovered, and for each one gives an example sequence of state transitions that reaches it.

An early paper [41] described how to model the behavior of certain types of public key protocols in terms of the action of the protocol plus participant and intruder knowledge, specify precisely the meaning of "security fault," and search the implicit state-space for such faults. Limitations on protocols that could be so analyzed have been relaxed (e.g., [42], [43], [44]).

Model-checking is also being used to analyze computer programs for security flaws [45]. The fundamental data structure is the program's control flow graph, and the fundamental concept being analyzed is a set of program properties of an execution path, as it evolves in accordance with the control flow graph. [46] reports success in analyzing large-sized well known software packages (Apache HTTPD, BIND, Postfix, OpenSSH, Samba, Sendmail) for a set of particular security flaws. Actions a program execution might take to exercise a flaw are described with finite state automata; the model-checking involves analyzing the control-flow graph to determine whether the program satisfies the transitions described by any fault automata.

Model-checking approaches are also finding application in modeling attacks on systems (e.g., [47], [12]), where the network state includes a description of hosts and their vulnerabilities and a description of connectivity. In that approach, an attacker's state includes capabilities and access gained so far in the course of an attack. A state transition occurs when there is a match between a capability in the attacker's state and a vulnerability in the network state, resulting (usually) in increased access somewhere in the network. States that represent an attacker's access to network assets (e.g., gaining root access on a host, or an ability to retrieve critical information from a database) reflect successful exploits. For every asset, one can ask whether it can be compromised (a precise definition of which depends on the model), and in principle determine the number of paths to states in which the asset is first compromised.

The difficulty, of course, is the size of the state space. Advances in state representation have led to an ability to represent very large state spaces; for example, [48] indicates that extremely large state-transition-rate diagrams can be represented compactly using symbolic data structures such as Binary Decision Diagrams (BDDs) and Multi-terminal Decision Diagrams (MDDs) (e.g., [26], [49]). These techniques have been shown to be able to represent state-transition-rate diagrams with 10^{20} or more states. Sheyner et. al. [12] report a model build-time of 2 hours on a model that has 229 bits of state. From the point of view of tractable formal models, 229 bits is remarkable; however, this serves only to blunt the onset of the curse of dimensionality. With respect to the need to represent the complexities of real systems, it is still very small. A nascent effort at finessing the state space issue by *sampling* paths through the state-space is reported in [50], in which a model with 1700 bits of state is analyzed. The fundamental idea is to use importance sampling to guide the path sampling strategy, in order to estimate inherent system security metrics such as the number of unique exploits that compromise a given asset.

C. State-Based Stochastic Methods

Combinatorial methods are quite limited in the stochastic behavior that they can express. While attack-tree analysis has become a staple in the diet of system security analysts, the classical formulation does not capture the dependence of security vulnerabilities on *sequencing* of events; to be successful, a buffer overflow attack that gains root access must precede the attack `rm -r *`.

State-space methods are much more comprehensive. They allow explicit modeling of complex relationships (e.g., [51]), and their transition structure encodes important sequencing information. Historically, state-space methods have been explored in the context of mathematical models that specify probabilistic assumptions about time durations and transition behavior. We now review those models and comment on how they are being applied in the security context.

1) Markov Reward Models [17], [8], [52]: Let $\{X(t), t \geq 0\}$ be a homogeneous finite state continuous time Markov chain (CTMC) with state space S and infinitesimal generator matrix $\mathbf{Q} = [q_{ij}]$. Let $P_i(t) = P\{X(t) = i\}$ denote the unconditional probability that the CTMC will be in state i at time t, and the row vector $\mathbf{P}(t) = [P_1(t), P_2(t), \ldots, P_n(t)]$ represent the transient state probability vector of the CTMC. The transient behavior of the CTMC can be described by the

Kolmogorov differential equation:

$$\frac{d\mathbf{P}(t)}{dt} = \mathbf{P}(t)\,\mathbf{Q}, \qquad \text{given } \mathbf{P}(0) \tag{1}$$

where $\mathbf{P}(0)$ represents the initial probability vector (at time $t = 0$). The steady-state probability vector $\pi = \lim_{t\to\infty} \mathbf{P}(t)$ satisfies:

$$\pi\mathbf{Q} = 0, \qquad \textstyle\sum_{i\in S} \pi_i = 1. \tag{2}$$

In addition to transient state probabilities, cumulative probabilities are sometimes of interest. Define $\mathbf{L}(t) = \int_0^t \mathbf{P}(u)\,du$; then $L_i(t)$ denotes the expected total time the CTMC spends in state i during the interval $[0, t)$. $\mathbf{L}(t)$ satisfies the differential equation:

$$\frac{d\mathbf{L}(t)}{dt} = \mathbf{L}(t)\,\mathbf{Q} + \mathbf{P}(0), \qquad \mathbf{L}(0) = 0. \tag{3}$$

With these definitions, most interesting dependability measures can be defined.

CTMCs with absorbing states deserve additional attention. Here, the measures of interest are based on the time a CTMC spends in non-absorbing states before an absorbing state is ultimately reached. To compute this measure, the state space $S = A \cup T$ is partitioned into the set A of absorbing states and the set T of non-absorbing (transient) states. Let $\mathbf{Q}_T$ be the submatrix of $\mathbf{Q}$ corresponding to the transitions between transient states. Then the time spent in transient states before absorption can be calculated by $\mathbf{L}_T(\infty) = \lim_{t\to\infty} \mathbf{L}_T(t)$ restricted to the states of the set T. The mean time to absorption (MTTA) can be written as $MTTA = \sum_{i\in T} L_i(\infty)$.

Assigning rewards to states or to transitions between states of a CTMC defines a Markov reward model (MRM). Rewards are referred to as *rate rewards* in the former case, and as *impulse rewards* in the latter case. If we consider rate rewards only, let the reward rate r_i be assigned to state i. Then, the random variable $Z(t) = r_{X(t)}$ refers to the instantaneous reward rate of the MRM at time t. The accumulated reward over the interval $[0, t)$ is given by

$$Y(t) = \int_0^t Z(u)\,du = \int_0^t r_{X(u)} du. \tag{4}$$

Based on the definitions of $X(t)$, $Z(t)$, and $Y(t)$, which are non-independent random variables, various measures can be defined. The most general is the distribution of the accumulated reward over time $[0, t)$, that is, $P\{Y(t) \le y\}$, which is difficult to compute for unrestricted models and reward structures (see [52], [53] for a survey of methods to compute the distribution of reward accumulated over a finite interval).

The problem is considerably simplified if we restrict ourselves to the expectations and other moments of random variables. In that case, the expected instantaneous reward rate can be computed from

$$E[Z(t)] = \sum_{i\in S} r_i\, P_i(t) \tag{5}$$

and the expected reward rate in steady-state (when the underlying CTMC is ergodic) is

$$E[Z] = \sum_{i\in S} r_i\, \pi_i. \tag{6}$$

To compute the expected accumulated reward over $(0, t)$ we use

$$E[Y(t)] = \sum_{i\in S} r_i\, L_i(t). \tag{7}$$

For models with absorbing states, the limit as $t \to \infty$ of the expected accumulated reward is called the *expected accumulated reward until absorption,* which is

$$E[Y(\infty)] = \sum_{i\in T} r_i\, L_i(\infty). \tag{8}$$

Note that the reward rate assignments (or reward structure) clearly depend on which attribute we are interested in with respect to a system's dependability and security. Markov and Markov reward models have been extensively used for dependability analysis of hardware systems (see, for example, [8], [9], [22]), real-time system performance in the presence of failures [54], [55], architecture-based analysis software systems (e.g., [56]), combined analysis of hardware-software reliability (e.g., [57], [58]), system performance analysis (e.g. [17]) and performability analysis (e.g., [8], [52], [59], [60], [61], [53]). Many sources for solution algorithms are available (e.g., [62], [63]), and many software packages exist (e.g., [62], [22], [64], [65], [66]). Several models constructed from and validated against measurement data have also been published [67], [68].

For complex systems with large numbers of components, the number of system states can grow prohibitively large. This is called the *largeness* problem for MRM models. Thus, significant work is being done to reduce the size of the Markov chain required for realistic system models. There are two general approaches for dealing with the size problem: *largeness avoidance* and *largeness tolerance* [22].

2) Largeness Avoidance Techniques: When largeness avoidance is employed, the size of a model is reduced and therefore a large model is not generated. State truncation methods [69], [70], hierarchical model solution [22], fixed point iteration [71], [72], and hybrid models that judiciously combine different model types [22] are examples of largeness avoidance.

State lumping (e.g., [73], [74]) has also been used extensively as an avoidance approach. Lumping reduces the size of a CTMC by considering the quotient of the CTMC with respect to an equivalence relation (i.e., replaces a set of states with a single lumped state) that preserves the Markov property and supports the desired performance measures defined on the CTMC. By solving the smaller CTMC, it is possible to compute exact results for the larger CTMC, and therefore measures of interest for the original model.

A *state-level* lumping technique is a lumping technique that exploits the lumping properties at the CTMC level. The main advantage of state-level lumping techniques is that they generate the optimal (i.e., smallest possible) lumped CTMC. However, since they can perform efficiently only on a sparse matrix representation of a CTMC, they have prohibitive space requirements for very large CTMCs; therefore, they are usually used along with other CTMC solution techniques. Buchholz [75] gives a state-level lumping algorithm with $O(mn)$ time complexity and $O(m+n)$ space complexity for computing the optimal (i.e., coarsest) lumping of a CTMC represented as a sparse matrix, where n is the number of states and m is the number of transitions of the CTMC.

Several authors have also addressed the problem of computing bisimilarity [76], which is, in some ways, similar to the problem of state-level CTMC lumping. Kanellakis and Smolka gave a partition refinement algorithm with time complexity $O(mn)$ [77]. They conjectured that an algorithm exists that reduces the time complexity to $O(m \log n)$. A few years later, Paige and Tarjan designed such an algorithm [78]. An implementation of Paige and Tarjan's algorithm can be found in [79]. Supposedly based on [78], $O(m \log n)$ complexity has been claimed without proof by Bernardo and Gorrieri [80] for CTMCs and by Huynh and Tian [81] for discrete-time Markov chains (DTMCs). Derisavi et al. [82] recently provided an $O(m \log n)$ lumping algorithm, along with a rigorous proof. The approach they take is based on Tarjan and Paige, and uses new data structures to obtain the bound.

In contrast, *model-level* lumping techniques identify appropriate lumping properties by operating on a higher-level formalism (see the following section for a description of one such formalism) and directly constructing a lumped CTMC, rather than by constructing the unlumped CTMC and then operating on it. The lumping equivalence relation is established by the modeling formalism itself in some model-level lumping techniques. That holds for stochastic well-formed nets (SWNs) [83] and replicate/join operators in stochastic activity network-based composed models (SANs) [84], in which the lumping results from equivalence of the replicas of a particular submodel. Extending the work of [84], Obal developed [85] a graph composition formalism and used the symmetry detection, a type of model-level lumping technique. The technique automatically identifies and exploits all the structural symmetries due to the interaction between submodels of a state-sharing composed model, that is, a model consisting of submodels that share a subset of their state variables. Restricted versions of a symmetry detection technique similar to the one described in [84] have also been used for process algebras in [86], [87]. Unlike state-level lumping, model-level lumping techniques do not always find the optimal lumping, because they do not operate at the CTMC level.

Other lumping techniques, which we call *compositional lumping* techniques, can be applied to composed models provided that the specific high-level formalism satisfies a particular set of assumptions. In these techniques, each of the individual interacting submodels is lumped separately from the others using a state-level lumping algorithm, and is then replaced in the overall model by its lumped version. For example, based on the fact that lumping is a congruence with respect to parallel composition in a number of process algebra formalisms and stochastic automata networks, compositional lumping can be used in those formalisms to generate lumped state spaces (e.g., [88], [89], [90], [91], [92], [93]). Most of the work on compositional lumping applies only to state-level lumping inside the submodels. In some cases, in addition to lumpability in each of the submodels, the structural symmetry of the interaction among the submodels may also be exploited to achieve even smaller CTMCs. In other words, for some composed models, a compositional lumping algorithm that applies the state-level and model-level lumping techniques at the same time could give an extra opportunity to shrink the CTMC. Therefore, a fairly general algorithm that integrates the two techniques for a compositional formalism is desirable.

Another largeness avoidance technique is called *aggregation*[1]. In this approach, as in lumping, a set of conditions for partitioning the set of states of a CTMC is given such that a smaller CTMC is constructed by replacing the set of states in each block of the partition with a single state. The aggregation differs from lumping in that the solution of the aggregated CTMC gives approximate results (with or without bounds) on the original CTMC, as opposed to the exact results that would be obtained from the lumped CTMC. Moreover, some aggregation techniques are only applicable to CTMCs that satisfy a strict set of conditions. However, aggregation conditions could result in a coarser partition, and therefore a smaller CTMC, compared to the lumping conditions. An aggregation technique for steady-state analysis of a general CTMC has been proposed in [94], [95]. It gives the best known bounds on the result but is computationally costly, and the bounds are tight only if the matrix satisfies some strict constraints. Bobbio and Trivedi [96], [97] extended Courtois's technique to the domain of transient analysis of CTMCs. Daly et al. [98] give a more general aggregation technique that can solve for both steady-state and transient measures of general CTMCs. It introduces a new partial order on the set of states of a CTMC that is a generalization of the concept of lumping.

3) Largeness Tolerance Techniques: Even a lumped CTMC can be extremely large, and hence *largeness tolerance* techniques are needed to provide practical modeling support for large CTMCs. In those techniques, one starts with a concise high-level representation of the system being modeled (usually a variant of stochastic Petri net or stochastic process algebra; see a following subsection). Then, new algorithms are designed to manipulate the large underlying CTMCs, and special data structures and/or representations are utilized to reduce the space requirements of the state space, the generator matrix, and the iteration vectors. Those techniques are usually, but not always, associated with compositional modeling.

Binary [26] and multi-valued decision diagram [99] (BDD and MDD) data structures have been successfully applied to efficiently explore and represent large unlumped state spaces. The key idea is to encode states as paths in a directed acyclic graph. Techniques that generate state spaces using decision diagrams are referred to as *symbolic* state-space exploration and representation techniques (e.g., [49], [100]).

MDD data structures have been used in [48] to explore large state spaces of models built using an *action synchronization* (a.k.a. *action-sharing*) high-level compositional formalism in which submodels interact by synchronized firing of a subset of their actions. Saturation, a state-space exploration technique that was introduced in [101], improved the running time of the algorithm given in [48] by up to a few orders of magnitude, thus enabling the exploration of even larger state spaces. In

[1]Note that different authors use the terms *lumping* and *aggregation* differently.

[48], [101], it was assumed that the state spaces of individual submodels were known a priori, i.e., the state spaces of the submodels computed by exploring the submodels in isolation are finite and result in the same state spaces they would have if they had been explored in interaction with the rest of the model. This assumption was relaxed later in [102]. Those state space exploration techniques are applicable to action-synchronization composed models that conform to a set of particular structural restrictions called the *logical product form* property [48].

One approach in space-efficient representation of generator matrices is to follow a divide-and-conquer strategy and represent the matrix with a set of relatively small component matrices that are appropriately combined. The earliest attempt using that approach was made by Plateau [103], [104], who proposed a technique in which the generator matrix of a CTMC generated from a specific compositional high-level formalism need not be explicitly stored. Instead, the matrix is implicitly represented as a mathematical expression consisting of *Kronecker* operators and a number of relatively small matrices derived from the structure of submodels. Later, the "Kronecker representation" technique was extended to more general formalisms, and a number of its shortcomings were resolved [105], [106], [107], [108], [109], [110]. The approach is applicable only to action-synchronization models that satisfy certain structural constraints.

A parallel effort was undertaken by Miner and Ciardo [111], who proposed the matrix diagram (MD) data structure to store the generator matrix of action-synchronization composed models. An MD is structurally similar to an MDD and, along with an MDD, represents the set of states and transitions of a very large CTMC. Efficient algorithms to manipulate MDDs and MDs have been given in [111]. In [111], the algorithm that generates the MD data structure is time-efficient, but works only for composed models that hold the logical product form property. Later, Miner [112] developed canonical MDs (CMDs), a proper subset of MDs, and presented an algorithm to store virtually any matrix in the form of a CMD. In particular, he used the algorithm to generate the CMD representation of the generator matrix of models based on the generalized stochastic Petri net (GSPN) formalism, which is a fairly general Markov modeling formalism. Since the algorithm added matrix elements to the CMD data structure one by one, and without exploiting any structural information, its running time was prohibitive. As mentioned above, both CMDs and MDs can represent virtually any generator matrix regardless of the modeling formalism from which it was generated. The challenge is then to develop algorithms that build (C)MD representations of generator matrices of other formalisms in a time-efficient manner.

In contrast, the *disk-based* approach first introduced in [113] performs steady-state solution by storing the generator matrix of the CTMC in the disk instead of the memory while using a variant of block Gauss-Seidel as the iterative solution algorithm. To increase the utilization of the CPU, the algorithm implementation concurrently fetches parts of the matrix from the disk and performs computation on other in-memory parts of the matrix. By using disk instead of memory to store the matrix, the technique enables the solution of CTMCs that are one or two orders of magnitude larger than what would be possible if using only memory.

Likewise, the "on-the-fly" technique of [114] completely avoids the storage of the generator matrix by (re)generating the elements of the matrix as they are needed in an iterative solution algorithm (steady state or transient solution). The elements are computed on-the-fly from the model, which is given in a high-level formalism. Repetitive calculations of the elements incur a substantial computational overhead.

The path-based approach is yet another largeness tolerance technique for performing transient analysis of CTMCs while avoiding the storage of the CTMC and possibly the iteration vector. In this approach, a limited number of paths (i.e., sequences of transitions) of the CTMC that make a major contribution toward the measures of interest are enumerated. Then, the reward is computed only for those paths. The first notable work based on that approach was given by de Souza e Silva and Gail [115], and they later improved it in [116]. Later, Qureshi improved the numerical stability and computational complexity of [116] in [117]. Most recently, Lam et al. [118] use Kronecker operators to represent both the CTMC and the iteration vector to compute approximate results for transient analysis of an action-synchronization composed model.

4) Other Challenges: When there is a large difference between failure and repair rates or failure and job arrival rates in the model, it leads to the *stiffness* problem for MRM models. Stiffness can be reduced by separating the performance and availability models, but the stiffness within the availability model remains. Stiffness may be avoided by using aggregation [96], [97] that yields approximate solutions. To tolerate stiffness, special stable stiff solvers may be used (e.g., [119], [120], [121], [122]).

Largeness and stiffness problems may also be caused by combining the performance and availability models in a single monolithic performability model. Solving an overall model of system behavior can potentially yield more accurate results than solving two smaller, less stiff models that only lead to approximate solutions. However, we should note that numerical difficulties arising from largeness and stiffness may very well negate this gain [52], [60].

A major objection to the use of homogeneous Markov models in the evaluation of performance and dependability behavior of systems is the assumption that the sojourn (holding) time in any state is exponentially distributed.

The exponential distribution has many useful properties that lead to analytic tractability, but it does not always realistically represent the observed distribution functions. One way to deal with non-exponential distributions is the phase-type approximation, which consists of modeling a distribution with a set of states and transitions between those states such that the holding time in each state is exponentially distributed [8], [123]. The simplest examples of phase approximation are the hyperexponential distribution with a coefficient of variation larger than 1, and hypoexponential distribution with a coefficient of variation less than 1. Although the method of phase-type approximation enables us to use MRMs, its major drawback is that it usually results in a large state space.

If transition rates in a CTMC are allowed to be time-dependent, where time is measured from the beginning of system operation, the model becomes a non-homogeneous CTMC. Such models are used in software reliability modeling (e.g., [124], [125]) and in hardware reliability models of non-repairable systems (e.g., [126]).

Due to the assumptions that holding times in the state are exponentially distributed and that past behavior of the process is completely summarized by the current state of the process, any observation instant in a homogeneous CTMC acts as a regeneration point for the process. The first assumption can be relaxed by allowing the holding time to have any distribution, thus resulting in a *semi-Markov process* (SMP), where the epoch of each state transition is a regeneration point [127]. This assumption can be relaxed by assuming that not all state transitions are regeneration points, thereby resulting in a *Markov regenerative process* (MRGP) [128].

5) Higher-Level Model Representations: CTMCs are rarely used directly to specify a system's model in a typical modeling process. Many high-level modeling formalisms have been created to fill the gap between CTMC specification and system design specification. Examples of those formalisms include variants of stochastic Petri nets (e.g., [129], [64], [130], [131]), variants of stochastic process algebras [132], [90], [80], [133], and interactive Markov chains (IMC) [93]. To illustrate the usefulness of these model types, we describe stochastic Petri nets and extensions in more detail, and illustrate their use in the context of security.

Stochastic Petri nets (SPNs) and extensions have been developed as extensions to un-timed Petri nets (originally introduced by C. A. Petri in 1962) with timed transitions for which the firing time distributions are assumed to be exponential. SPNs have been extensively used in the area of dependability evaluation (e.g., [8], [134]) due to the small size of their descriptions and their visual/conceptual clarity. They allow the designer to focus more on the system being modeled rather than on error-prone and tedious manual creation of a lower-level MRM.

To specify an SPN, one has to define a set of places P, a set of transitions T, and a set A of arcs between transitions and places: $A \subseteq (P \times T) \cup (T \times P)$. Each place can contain zero or more tokens. Graphically, places are depicted as circles, transitions as bars, tokens as dots in circles, and arcs as arrows.

The distribution of tokens over the places is called a *marking* and corresponds to the notion of state in a Markov chain. All places from which arcs go to a particular transition are called the *input places* of that transition. All places to which arcs go from a particular transition are called the *output places* of the transition. A transition is said to be *enabled* when all of its input places contain at least one token. If a transition is enabled it may fire. Upon firing, a transition removes one token from each of its input places and puts one token in each of its output places, possibly causing a change of marking, i.e., a change of state.

The firing of transitions is assumed to take an exponentially distributed amount of time. Given the initial marking of an SPN, all the markings as well as the transition rates can be derived, under the condition that the number of tokens in every place is bounded. Thus a Markov chain is obtained.

Classically, SPNs and extensions are solved via an underlying MRM that can be automatically derived, thereby making it possible to use the wide variety of available techniques for MRMs.

In the last two decades many extensions to the basic SPN model have been proposed to enhance its modeling power and flexibility of use. They include arcs with multiplicity, a shorthand notation for multiple arcs between a place and a transition, immediate or instantaneous transitions that fire in zero time, and inhibitor arcs from places to transitions that prevent a transition from firing as long as there are tokens in the place. The most popular model of this type is called generalized stochastic Petri nets (GSPNs) [129]. More flexible firing rules have also been proposed, most notably the introduction of gates in stochastic activity networks (SANs) [130], [131] and guards or enabling functions in stochastic reward nets (SRNs) [64]. The extended stochastic Petri net (ESPN), in which general firing time distributions are allowed, has a semi-Markov process [135] as the underlying process, when certain restrictions are met.

Deterministic stochastic Petri nets (DSPNs) allow the definition of immediate, exponential, and deterministic transitions [136]. The stochastic process underlying a DSPN is a Markov regenerative process. Markov regenerative stochastic Petri nets (MRSPN) generalize DSPNs and still have MRGP as an underlying stochastic process [137]. Concurrent generalized Petri nets (CGPN) allow simultaneous enabling of any number of immediate, exponentially, and generally distributed timed transitions, provided that all generally distributed transitions are enabled at the same instant. The stochastic process underlying a CGPN is also an MRGP [138]. Fluid stochastic Petri nets allow for continuous state variables [139], [140].

D. Applications to Security Modeling

As described earlier, state-based techniques have been extensively developed and used in classical dependability contexts. They are now also beginning to be used in security analysis. For example, Ortalo et al. [141] have proposed modeling of known system vulnerabilities using "privilege graphs," followed by a combination of the privilege graphs with simple assumptions about attacker behavior to obtain "attack-state graphs." The latter can be analyzed using the Markovian reward models described above to obtain probabilistic measures of security. An interesting definition of reward used here is the "effort" needed to make a transition (which usually represents some sort of system compromise).

The types of Markovian models described above ascribe distributional properties to high-level system and attacker behaviors. Accepting this, we have still the issue of quantifying the scale of these stochastically modeled activities, particularly those related to attacks. Early ground-breaking work in this regard was done by Littlewood et al. [142]. Their work was exploratory in nature and identified "effort" made by an attacker as an appropriate measure of the security of the system. Effectively, the model consisted of only two states, viz. "working" and "security failed state." The latter state was assumed to be an absorbing state. With respect to the above discussion, the relevant security measure turns out to be “mean effort to (security) failure.” Jonsson and Olovsson [143] attempted to build a quantitative Markov model of attacker behavior using data from several experiments they conducted over a two-year period. They postulated that the process representing an attacker may be broken into multiple phases, each of which has an exponential time distribution. The overall attacker behavior therefore requires non-exponential characterization.

More recently, Singh et al. [13] have used stochastic activity networks to validate an intrusion-tolerant system, emphasizing the effects of intrusions on the system behavior and the ability of the intrusion-tolerant mechanisms to handle those effects, while using very simple assumptions about the discovery and exploitation of vulnerabilities by the attackers to achieve those intrusions. Gupta et al. [14] have used a similar approach to evaluate the security and performance of several intrusion-tolerant server architectures. Madan et al. [10] have used a semi-Markov model to evaluate the security properties of an intrusion-tolerant system. Depending on the particular attack scenario, various states may be associated with failure of availability, integrity, and confidentiality. The security may then be quantified in terms of the mean time to security failure and in terms of the absorption probabilities.

Furthermore, Dacier et al. [144] has proposed a two-stage technique that starts by converting a privilege graph into an SPN by treating each atomic attack as a transition in a stochastic Petri net. In the second stage, the markings of this SPN generate a continuous-time Markov chain. Making simple assumptions about the attacker, the chain may be analyzed using the Markov reward techniques to obtain probabilistic measures of security in terms of the mean time to reach “security failed” states and other related measures. In [145], Wang et al. use stochastic reward nets to model both attacker and system behavior for an intrusion-tolerant architecture named SITAR [4]. Likewise, probabilistic methods have been used to model the DPASA [21] architecture. In the DPASA project, system validators combined structured requirement specification, probabilistic modeling, experimental evaluation, logical arguments, and formal methods to build an overall survivability argument for the architecture. Figure 3 shows an example of an “argument graph” that links arguments that make use of these methods together.

A promising application of stochastic state-based methods in the security context is to quantify direct, high-level measures of the *service* that one can expect from a computer system or network in spite of cyber attacks that may occur. In order to do so, one needs a notion of time in the system model and a probabilistic notion of system and attacker behavior. High-level service measures can be defined to quantify system performance under cyber attack, so that 1) systems can be represented as state-level models in a way that captures either known or unknown vulnerabilities, 2) attacker behavior can be modeled in such frameworks, and 3) measurement can be used to quantify parameters describing known vulnerabilities.

A model for probabilistic validation of security with respect to high-level system properties (e.g., availability, privacy, and

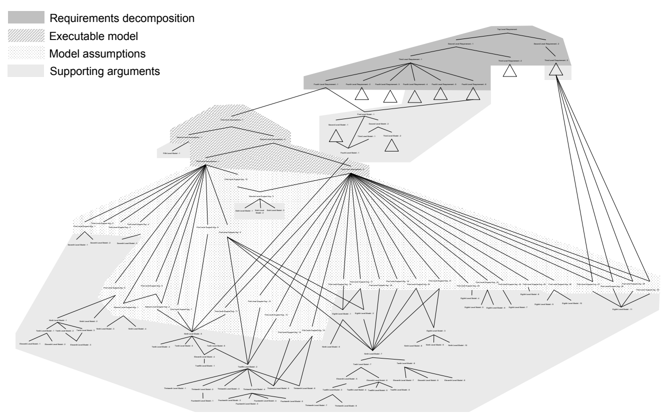

Fig. 3. DPASA Argument Graph Structure

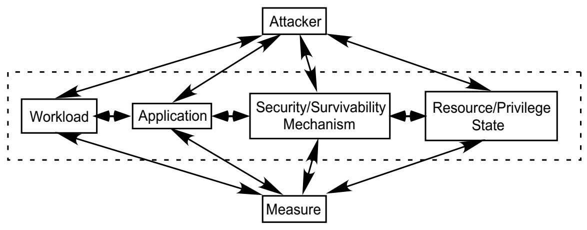

Fig. 4. Probabilistic security model structure

integrity) should have several components. It should contain representations of attackers, the system, and the assets, resources, and privileges associated with the system. It should represent attacker decision-making and all temporal aspects of his, and the system’s, behavior, and the application (or applications) that provide services of interest [146].

Model specification of behaviors needs to focus on the appropriate stochastic measures and their relationship to services, and determine how they are reflected in the model. Such a model ought to capture relevant *attacker behavior*, the *workload* demanded of the system, the intended *application* (which defines the service that must be provided in spite of cyber attacks that may occur), a model of *security and survivability mechanisms* employed, and a model of the *resources/privileges* that are needed by the application to provide its services. Figure 4 shows the relationship of these parts of the model to one another. The arcs connecting submodels in the figure represent possible interactions between submodels that can change their state. For example, the attacker may be able to change the state of a resource or amount of privilege granted to him (as represented by the directed arc from the attacker to the Resource/Privilege State submodel), or the attacker may change his state (and hence change his behavior) by using knowledge he has gained by observing the state of the system (represented by the directed arcs from the system submodels to the attacker.) This general framework [146] is one we are using to develop models of specific applications on specific architectures.

Development of measurement techniques is just as important as the development of models to quantify security. Measurements should be developed for two purposes in this regard: 1) to guide construction of and provide input parameter values for models, and 2) to validate the correctness of models. In particular, with regard to the first objective, the appropriate level of detail/abstraction depends on the input parameter values (obtained from measurement data) available for each model. The type and accuracy of input parameter values available will depend on the stage of development of the system that is being validated. For existing systems, one could use methods similar to Jonsson's as a starting point to obtain values that quantify the behavior of an attacker. A recent study that placed significant emphasis on the development of an attacker effect model is [21]. The attacker model developed there is quite detailed, but tailored to the system being studied. It's not clear if it will be possible to build attacker models that are more generic, but still have the detail needed for meaningful representation.

Concerning the Resource/Privilege State model, one could use a security scanner like Nessus [147] combined with a network exploration and security auditing tool like nmap [148] to obtain model parameter values. Likewise, tools like COPS [149], Tiger [150], or Ferret [151] could be used to quantify host security vulnerabilities. The advantage of these tools is that the list of vulnerabilities is updated as new vulnerabilities are discovered. The application, security and survivability mechanism, and workload models are more case-specific, and require further study to determine appropriate input parameters and their values. It appears that the best route forward is from specific to general. We and others are currently building models of this type for specific system designs, and investigating how to obtain parameter values for them. It is our hope that these experiences will guide us in constructing more general models and approaches for parameter value estimation.

The second objective, model validation, is a more difficult issue, both for classical dependability models and for security models whose goal is quantification. We are not aware of any "silver bullet," other than the hard and meticulous work and significant time required to collect data on attackers and systems that are intended to be secure. Once that has been done, one can compare the insights gained from the data collected to those obtained from quantitative models. Note, however, that the value of most quantitative security models, like most classical dependability models, will be in gaining insight and making decisions about how best to make a system secure/dependable, rather than making a precise statement about a specific system's absolute dependability or security.

E. Simulation

As just argued, the state space of a system model may be too large to be analyzed in its entirety. Nevertheless, in principle we can construct statistical estimators of all the system measures computed by the quantitative techniques we have described. Rather than generate and analyze the entire state space, we generate and analyze randomly chosen paths through the state space; typically, we sample many paths independently of each other (in a strict probabilistic sense of "independent"). This focus on evaluating a system by individual trajectories of the system is commonly known as *simulation.*

1) Statistical Issues: A major issue when using simulation to estimate system measures is that of how to do so in a way that ensures the statistical quality of the estimates. The first concern is that the estimator be *unbiased*, which is a technical term that means that the mean value of the estimator (itself a random variable) is the same as the mean value of the random system measure being estimated. Proof that an estimator is unbiased is powerful insofar as it implies (by the law of large numbers) that the arithmetic average of many independent unbiased estimates converges to the mean of the random samples, which is the same as the mean of the system measure being estimated.

A more subtle statistical issue is the variance exhibited by the estimator. To illustrate this issue, imagine a system measure whose mean is a small probability, say $1e-5$. Further imagine that one way of estimating this probability creates estimates that fall almost entirely in the range $0.95e-5$ to $1.05e-5$, and another generates estimates that fall almost entirely in the range $1e-7$ to $1e-3$. The first method of estimation is clearly better, because its values are almost always closer to the true mean than the values of the second method are. The practical impact of using a low variance estimator is that for a given level of accuracy, lower variance implies that fewer samples are needed to achieve that accuracy.

The issues of bias and variance are particularly significant when the measure of interest is a small probability, and therefore are significant in the context of dependability and security. So-called *variance reduction techniques* [152] have been developed in reliability and performance analysis, but are characterized by the need to exploit a model's structure in order to be provably effective. Work is needed to explore how variance reduction technology can be applied to models focused on the idiosyncrasies of security issues.

The technique of simulating a system independently many times is necessary when one is interested in transient measures; the measures of interest can be repeatedly sampled at the given instants of interest. This same technique is used to estimate asymptotic measures. For example, consider the estimation of a queue's asymptotic average length. Using independent replications one runs N statistically independent simulations, and generates N independent samples of the average queue length. Each sample is the queue length averaged over an interval of simulation time $[s, T]$, where the system is deemed to be in "steady state" by time s, and T is the simulation termination time. Standard statistical methods can be used to compute confidence intervals around a sample mean. However, replications per se are sometimes not needed to estimate asymptotic measures. One can instead push the model along a single sample path, but for a period of simulation time long enough to force the system into steady state, and then use samples taken across time, rather than samples taken across replications. Applied to the example above, one runs the simulation to time T'. The method of batch means [153] partitions the interval $[s, T']$ into successive epochs of duration Δ, and measures average queue length over each. For many types of models, when Δ is large, the measurements are nearly independent, so confidence intervals can be constructed. Depending on 1) how long the simulation must be run until reaching steady state (i.e., time s), 2) the length of simulation time needed for a "good" sample ($T - s$), and 3) the size of the epoch needed to give near-statistical independence to successive estimates (i.e., Δ), the computational cost of doing a long run can be smaller than the computational cost of N independent replications. If we take the total length of simulation time traversed as an indicator of computational work, the cost of N independent replications is $N \times (s + (T - s)$, while the cost of a long run is $s + N \times \Delta$. A little algebra shows that independent replications cost more when $(N - 1)s > N(\Delta - (T - s))$.

The point of selecting a large Δ is to try to heuristically create statistical independence between successive samples in a long run. It is possible to bring more rigor to this intuition, using the notion of regeneration points [154]. The idea is that certain systems sometimes enter states from which they probabilistically start over in the sense that the future behavior is independent of the past. The simplest example of such a regeneration point is the beginning of an idle period for an M/G/1 queue; the Poisson arrivals imply that there is no probabilistic memory of the time since the last arrival, and the fact that the queue is empty implies that there are no complications owing to interrupted service times. At such an instant the future behavior of the queue is in a probabilistic sense completely independent of anything that has happened in the past. The implication is that epochs separated by regeneration points can, for the purposes of sampling measures, serve as independent replications of system behavior. Just as discovery of sampling schemes that lead to variance reduction depends very much on the particulars of the model, so likewise does identification of regeneration points. To make this technology work in the context of security analysis, we must identify or construct models in which regeneration points can be readily identified.

The challenges of using stochastic simulation models on system models in a security context partition along the lines of attributes of interest. Those attributes described by numbers, e.g., availability or performability, can be treated using known quantitative techniques; however, optimizations such as variance reduction require domain-specific insight that has yet to be generally developed in this context. New attributes more closely tied to security (e.g., confidentiality, non-repudiation, and authentication) are fundamentally different in that they are system properties, not system measures. If classical stochastic simulation output analysis is to be used to help evaluate these attributes, then numeric measures of some sort will have to be developed to quantify these properties.

2) Simulation Model Representation: Model formalisms are often developed to expose underlying structure common to different models. The beauty of a stochastic Petri net formalism is that one may use it equally well to describe a communication synchronization protocol and a machine maintenance and repair schedule, since the underlying mathematics and analysis algorithms remain the same. The flip side of the coin, however, is that such formalisms have built-in constraints whose effects limit how a model can be expressed. Some models just don't fit the formalisms. A common use of simulation is to express systems using formalisms that are not as mathematically tractable as Markov chains or SPNs, but which allow greater freedom of expression, and "look" more like the systems they represent. Simulation languages have been developed to this end (for a comprehensive overview of simulation languages, see [155]); general-purpose programming languages are also used, e.g., [156], [157]. Generality of expression is extremely important if we are to capture the complicated unexpected interactions that trigger security failures.

On the other hand, certain types of simulation models demand less detail, rather than more. A good example of this is Internet worm propagation, for which differential equations have been used [158], [159], [160], [161], [162]. The need for such high-level abstractions is computational; worms propagate over the entire Internet, and it is computationally infeasible to model individually infected hosts, sending individual probes and infection packets.

3) Simulation and Security Analysis Today: By far the most common use of simulation today in a security context is the use of normal network simulation tools to model a system, and then model traffic representing attacks. Such models are useful for quantifying diverse system measures in the midst of attacks and counter-measures, e.g., see [163], [164], [165]. Equally common, though, is the use of simulation as a means of education and discovery. For example, there is an effort underway at the Naval Postgraduate School [166], [167] to build a simulator designed to get students engaged in the process of system configuration and cyber attack defenses, in addition to playing the role of cyber attackers. The interest here is less on the specifics of quantifying system behavior, and more on providing enough realism for students to learn about security issues. Similar efforts are underway elsewhere [168], [169].

Simulation is also being used to help government agencies practice appropriate responses to cyber attacks on their information technology infrastructure. Two notable examples are a cyber attack exercise conducted in Seattle in conjunction with the May 2003 TOPOFF dirty-bomb exercise [170], and the October Livewire cyber war exercise [171]. In the Livewire exercise, simulated attacks on a simulated network caused disruption of simulated services. Exercise players then worked through their responses (e.g., who to call, appropriate network reconfiguration, and so forth) to degraded capability. Again, the users looked to the simulation to provide a "realistic" simulated network behavior, not to quantify system metrics precisely (although capturing the general trend of system metrics under cyber attack is critical to the whole approach).

Despite the seeming divergence of this style of simulation from very mathematical quantitative analyses, there is potential for closer linkage. Loose requirements on the accuracy of quantitative measures open the possibility for other types of models and analyses to play an important role as fast approximations, albeit hidden from the user. We can look forward to work that incorporates diverse modeling methodologies in such applications of simulation.

IV. Challenges and Conclusions

In the previous sections, we have reviewed measures that are pertinent to dependability and security evaluation, surveyed existing techniques for dependability evaluation, and given examples of how those techniques are currently being applied to the evaluation of certain security properties. While these applications suggest that there is merit to using stochastic techniques to evaluate security properties, they also suggest that significant new work is necessary to create a sound, model-based framework for quantifying system security.

That goal is clearly important, since history suggests that it will be difficult, if not impossible, to build systems that can be shown to be perfectly secure. Hence, in order to have confidence that a given design will perform its intended function, we must be able to quantify its security. At the highest level, we believe that this work falls into two categories: 1) modeling attacker behavior, and 2) creating a single, comprehensive methodology for evaluating whether a design meets one or more high-level requirements related to security. We outline the issues and challenges related to each of these needs in the following.

The first challenge is related to appropriate modeling of the behavior of cyber attackers. Just as appropriate fault models are critical to dependability evaluation, appropriate attacker models are critical to quantitative security evaluation. Determining the appropriate level of detail/abstraction in an attacker model is very important, and depends on the scope and purpose of the model. Different attacker models will be needed for different purposes and different attack classes. For a given model, the level of detail/abstraction that is appropriate will depend on many factors. For example, the system sub-models should represent the parts of a system that are important, relative to the types of attacks considered and the expression of a particular security measure. In particular, they must be detailed enough to support the expression of those parts of state that an attacker may change and those parts that may change his behavior.

Depending on the nature of the attack, the attacker model may either represent details of the attack or intrusion itself (corresponding to explicit representation of a fault in a dependability model) or represent the effect of the intrusion (corresponding to the representation of the error in a dependability model). We believe that by representing attacker behavior in terms of effects, rather than attacks/intrusions, we can cover a large class of attacks/intrusions (including unknown (“zero day”) attacks) in a model.

Development of a comprehensive methodology for system-level security quantification is also a significant challenge. As described earlier in the paper, stochastic evaluation techniques originally intended for use in dependability evaluation have been successfully used to evaluate certain security attributes, including availability and survivability. However, other attributes, such as confidentiality and non-repudiation, are more difficult to evaluate using standard, stochastic, techniques.

These measures may be better validated via so-called “formal” methods. The different natures of these multiple security measures suggest that the individual application of any of the techniques we have described is insufficient to validate large systems that are intended to be secure. While each of those techniques has the ability to evaluate certain kinds of security measures, such abilities may have limitations when one is attempting to use a single approach to validate the system with respect to a fairly high-level security requirement.

What is needed is an integrated validation framework that permits the use of multiple evaluation techniques in an organized manner. Starting with a system and a high-level set of security requirements, the framework should provide a top-down approach to methodically break the problem of validating the system with respect to its security requirements into manageable tasks, and provide steps that deal with each of those tasks. Each step would use one or more individual evaluation techniques. A symbiotic relationship should be established among the various techniques such that they complement and supplement each other to build the overall argument. Such an approach would make it possible to handle the validation of very large-scale systems, producing systematic and well-documented arguments about their security. Such integrated approaches to evaluation have been applied in the safety community, resulting in so-called “safety cases,” suggesting that a similar approach might be useable for security quantification.

In summary, stochastic evaluation techniques inspired by dependability evaluation methods, have the potential to be used, with appropriate extension, for security evaluation. Several studies that take this approach have already been made, indicating the promise of this approach. However, there are still significant obstacles to the creation of a comprehensive, integrated approach to the evaluation of multiple security properties, as outlined above. There are ample opportunities for further research.

Acknowledgment

We would like to thank our past and current research sponsors for supporting this work, and our colleagues and students, both past and present, who also contributed to the research described herein. We would particularly like to thank Mr. Salem Derisavi, Mr. Yun Liu, Dr. Bharat Madan, and Mr. Dazhi Wang in this regard. We would also like to thank Ms. Jenny Applequist for her editorial assistance. Our sponsors include the the NSF (CCR-0209144, EIA-99-75019, INT-0233490, CCR-0311616, CNS-0406351), AFOSR MURI (F49620-1-0327), the Defense Advanced Research Projects Agency, Motorola, and Pioneer Hi-Bred. Our work was also supported under Award No. 2000-DT-CX-K001 from the Office for Domestic Preparedness, U.S. Department of Homeland Security. Points of view in this document are those of the author(s) and do not necessarily represent the official position of the U.S. Department of Homeland Security, or any of the other sponsors. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

[1] Y. Deswarte, L. Blain, and J. C. Fabre, “Intrusion tolerance in distributed computing systems,” in *Proceedings of the IEEE Symposium on Research in Security and Privacy*, May 1991, pp. 110–121.

[2] B. Dutertre, V. Crettaz, and V. Stavridou, “Intrusion-tolerant enclaves,” in *Proceedings of the IEEE International Symposium on Security and Privacy*, Oakland, CA, May 2002, pp. 216–224.

[3] M. Cukier, J. Lyons, P. Pandey, H. V. Ramasamy, W. H. Sanders, P. Pal, F. Webber, R. Schantz, J. Loyall, R. Watro, M. Atighetchi, and J. Gossett, “Intrusion tolerance approaches in ITUA,” in *Supplement of the 2001 International Conference on Dependable Systems and Networks*, Göteborg, Sweden, July 2001, pp. B–64–B–65.

[4] F. Wang, F. Gong, C. Sargor, K. Goševa-Popstojanova, K. S. Trivedi, and F. Jou, “SITAR: A Scalable Intrusion Tolerance Architecture for Distributed Services,” in *Proceedings of the IEEE 2nd SMC Information Assurance Workshop*, United States Military Academy, West Point, New York, June 5-6 2001, pp. 38–45.

[5] C. Landwehr, “Formal models for computer security,” *Computer Surveys*, vol. 13, no. 3, pp. 247–278, September 1981.

[6] J. Lowry, “An initial foray into understanding adversary planning and courses of action,” in *Proceedings of the DARPA Information Survivability Conference and Exposition II (DISCEX’01)*, 2001, pp. 123–133.

[7] A. Avizienis, J. Laprie, and B. Randell, “Fundamental concepts of dependability,” LAAS-CNRS, Tech. Rep. N01145, April 2001.

[8] K. S. Trivedi, *Probability and Statistics with Reliability, Queuing, and Computer Science Applications, 2nd Edition*. John Wiley and Sons, New York, 2001.

[9] M. L. Shooman, *Probabilistic Reliability: An Engineering Approach, 2nd Edition*. R. E. Krieger Publishing Co., Malabar, FL, 1990.

[10] B. Madan, K. Goševa-Popstojanova, K. Vaidyanathan, and K. Trivedi, “Modeling and quantification of security attributes of software systems,” in *Proc. Int. Conf. Dependable Systems and Networks*, 2002, pp. 505–514.

[11] S. Jha, O. Sheyner, and J. Wing, “Minimization and reliability analysis of attack graphs,” CMU-CS-2-109, Tech. Rep., May 2002.

[12] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. Wing, “Automated generation and analysis of attack graphs,” in *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, May 2002, pp. 273–284.

[13] S. Singh, M. Cukier, and W. H. Sanders, “Probabilistic validation of an intrusion-tolerant replication system,” in *Proc. Int. Conf. on Dependable Systems and Networks*, June 2003, pp. 616–624.

[14] V. Gupta, V. V. Lam, H. V. Ramasamy, W. H. Sanders, and S. Singh, “Dependability and performance evaluation of intrusion-tolerant server architectures,” in *Dependable Computing: Proc. of the First Latin-American Symposium (LADC 2003)*, ser. LNCS, vol. 2847, 2003, pp. 81–101.

[15] J. F. Meyer, “On evaluating the performability of degradable computing systems,” *IEEE Trans. Comput.*, vol. C-29, no. 8, pp. 720–731, August 1980.

[16] W. H. Sanders and J. F. Meyer, “A unified approach for specifying measures of performance, dependability, and performability,” in *Dependable Computing for Critical Applications, Vol. 4 of Dependable Computing and Fault-Tolerant Systems*, A. Avizienis, H. Kopetz, and J. Laprie, Eds. Springer-Verlag, 1991, pp. 215–237.

[17] G. Bolch, S. Greiner, H. de Meer, and K. S. Trivedi, *Queueing Networks and Markov Chains*. New York: John Wiley & Sons, 1998.

[18] R. J. Ellison, D. A. Fisher, R. C. Linger, H. F. Lipson, T. Longstaff, and N. R. Mead, “Survivable network systems: An emerging discipline,” CMU Software Engineering Institute, Tech. Rep. CMU/SEI-97-TR-013, November 1997.

[19] Y. Liu and K. S. Trivedi, “A general framework for network survivability quantification,” *to appear in Proc. 12th GI/ITG Conference on Measuring, Modelling and Evaluation of Computer and Communication Systems (MMB) together with 3rd Polish-German Teletraffic Symposium (PGTS)*, 2004.

[20] Y. Liu, V. B. Mendiratta, and K. S. Trivedi, “Survivability analysis of telephone access network,” in *Proc. IEEE Intl. Symposium on Software Engineering (ISSRE’04)*, 2004.

[21] F. Stevens, T. Courtney, S. Singh, A. Agbaria, J. F. Meyer, W. H. Sanders, and P. Pal, “Model-based validation of an intrusion-tolerant information system,” in *Proceedings of the 23rd Symposium on Reliable Distributed Systems (SRDS 2004)*, Florianpolis, Brazil, October 2004.

[22] R. A. Sahner, K. S. Trivedi, and A. Puliafito, *Performance and Reliability Analysis of Computer Systems: An Example-Based Approach Using the SHARPE Software Package*. Kluwer Academic Publishers, 1996.

[23] http://www.relexsoftware.com/products/relanalysissoft.asp.

[24] M. Malhotra and K. Trivedi, “Power-hierarchy of dependability model types,” *IEEE Transactions on Reliability*, vol. 43, no. 2, pp. 493–502, Sept. 1994.

[25] S. Rai, M. Veeraraghavan, and K. Trivedi, “A survey on efficient computation of reliability using disjoint products approach,” *Networks*, vol. 25, no. 3, pp. 147–163, 1995.

[26] R. E. Bryant, “Graph based algorithms for Boolean function manipulation,” *IEEE Transactions on Computers*, vol. C-35, no. 8, pp. 677–691.

[27] X. Zang, H. Sun, and K. Trivedi, “A BDD-based algorithm for reliability analysis of phased-mission systems,” *IEEE Transactions on Reliability*, vol. 48, no. 1, pp. 50–60, March 1999.

[28] J. E. Arsenault and J. A. Roberts, *Reliability and Maintainability of Electronic Systems*. Computer Science Press, Rockville, MD, 1980.

[29] R. E. Barlow and F. Proschan, *Statistical Theory of Reliability and Life Testing*. Holt, Rinehart and Winston, New York, 1975.

[30] B. S. Dhillon and C. Singh, *Engineering Reliability: New Techniques and Applications*. Wiley, New York, 1981.

[31] E. Henley and H. Kumamoto, *Reliability Engineering and Risk Assessment*. Prentice-Hall, Englewood Cliffs, NJ, 1981.

[32] N. G. Leveson, *Safeware: System Safety and Computers*. Addison-Wesley Publishing Co., 1995.

[33] J. B. Dugan and M. R. Lyu, “Dependability modeling for fault-tolerant software and systems,” in *Software Fault Tolerance*, M. R. Lyu, Ed. Chichester: John Wiley & Sons, 1995, pp. 109–138.

[34] J. B. Dugan, S. J. Bavuso, and M. A. Boyd, “Fault trees and sequence dependencies,” in *Proc. Reliability and Maintainability Symposium*, 1990, pp. 286–293.

[35] J. B. Dugan, “Fault trees and imperfect coverage,” *IEEE Transactions on Reliability*, vol. 38, no. 2, pp. 177–185, 1989.

[36] X. Zang, D. Wang, H. Sun, and K. Trivedi, “A BDD-based algorithm for analysis of multistate systems with multistate components,” *IEEE Transactions on Computers*, vol. 52, no. 12, pp. 1608–1618, December 2003.

[37] Y. Ma and K. Trivedi, “An algorithm for reliability analysis of phased-mission systems,” *Reliability Engineering and System Safety*, vol. 66, no. 2, pp. 157–170, 1999.

[38] “CAFTA: a fault tree analysis tool designed for PSA,” in *Proc. of Probabilistic Risk Assessment and Risk Management (PSA’87)*, vol. 2, Zurich, Switzerland, 1987, pp. 588–592.

[39] http://www.ds-s.com/risk_and_reliability_tools.asp.

[40] B. Schneier, *Secrets and Lies: Digital Security in a Networked World*. John Wiley & Sons, August 2000.

[41] D. Dolev and A. Yao, “On the security of public key protocols,” *IEEE Transactions on Information Theory*, vol. 29, no. 2, pp. 198–208, 1983.

[42] C. Meadows, “Applying formal methods to the analysis of a key management protocol,” *Journal of Computer Security*, vol. 1, no. 1, pp. 5–36, 1992.

[43] T. Woo and S. Lam, “A semantic model for authentication protocols,” in *Proc. of the 1993 IEEE Symposium on Security and Privacy*, 1993, pp. 178–195.

[44] W. Marrero, E. Clark, and S. Jha, “Modeling checking for security protocols,” Carnegie-Mellon University, Tech. Rep. CMU-SCS-97-139, May 1997.

[45] F. Besson, J. Jensen, D. L. Métayer, and T. Thorn, “Model checking security properties of control flow graphs,” *J. Computer Security*, vol. 9, no. 3, pp. 217–250, 2001.

[46] H. Chen, D. Dean, and D. Wagner, “Model checking one million lines of C code,” in *Proceedings of the 11th Annual Network and Distributed System Security Symposium*, San Diego, CA, 2004.

[47] R. W. Ritchey and P. Ammann, “Using model checking to analyze network vulnerabilities,” in *Proceedings of the IEEE Symposium on Security and Privacy*, May 2000, pp. 156–165.

[48] G. Ciardo and A. S. Miner, “Efficient reachability set generation and storage using decision diagrams,” in *Proc. 20th Int. Conf. Application and Theory of Petri Nets*, ser. LNCS, vol. 1639. Springer, 1999, pp. 6–25.

[49] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang, “Symbolic model checking: 10^{20} states and beyond,” *Information and Computation*, vol. 98, no. 2, pp. 142–170, June 1992.

[50] S. Singh, J. Lyons, and D. Nicol, “Fast model-based penetration testing,” in *Proc. of the 2004 Winter Simulation Conference*, Washington, DC, December 2004.

[51] J. K. Muppala, M. Malhotra, and K. S. Trivedi, “Markov dependability models of complex systems: Analysis techniques,” in *Reliability and Maintenance of Complex Systems*, S. Ozekici, Ed. Berlin, Germany: Springer, 1996, pp. 442–486.

[52] B. Haverkort, R. Marie, G. Rubino, and K. S. Trivedi, *Performability Modeling Tools and Techniques*. Chichester, England: John Wiley & Sons, 2001.

[53] K. S. Trivedi, J. K. Muppala, S. P. Woolet, and B. R. Haverkort, “Composite performance and dependability analysis,” *Performance Evaluation*, vol. 14, no. 3-4, pp. 197–215, 1992.

[54] J. K. Muppala, S. P. Woolet, and K. S. Trivedi, “Real-time systems performance in the presence of failures,” *IEEE Computer*, vol. 24, no. 5, pp. 37–47, May 1991.

[55] K. S. Trivedi, S. Ramani, and R. M. Fricks, “Recent advances in modeling response-time distributions in real-time systems,” *Proceedings of the IEEE*, vol. 91, no. 7, pp. 1023–1037, 2003.

[56] K. G. Popstojanova and K. Trivedi, “Architecture based approach to reliability assessment of software systems,” *Performance Evaluation*, vol. 45/2-3, pp. 179–204, 2001.

[57] S. Garg, Y. Huang, C. M. R. Kintala, S. Yajnik, and K. Trivedi, “Performance and reliability evaluation of passive replication schemes in application level fault tolerance,” in *Proc. Twenty-Ninth International Symposium on Fault-Tolerant Computing*, June 1999, pp. 322–328.

[58] J.-C. Laprie and K. Kanoun, “X-ware reliability and availability modeling,” *IEEE Trans. Software Eng.*, vol. 18, no. 2, pp. 130–147, 1992.

[59] W. H. Sanders and J. F. Meyer, “Performability evaluation of distributed systems using stochastic activity networks,” in *Proc. Int. Conf. on Petri Nets and Performance Models*, 1987, pp. 111–120.

[60] Y. Ma, J. Han, and K. Trivedi, “Composite performance & availability analysis of wireless communication networks,” *IEEE Transactions on Vehicular Technology*, vol. 50, no. 5, pp. 1216–1223, Sept. 2001.

[61] R. M. Smith, K. S. Trivedi, and A. V. Ramesh, “Performability analysis: Measures, an algorithm, and a case study,” *IEEE Trans. Comput.*, vol. C-37, no. 4, pp. 406–417, Apr. 1988.

[62] W. J. Stewart, *Introduction to the Numerical Solution of Markov Chains*. Princeton, 1994.

[63] A. Reibman, R. M. Smith, and K. Trivedi, “Markov and Markov reward models: A survey of numerical approaches,” *European Journal of Operations Research*, pp. 257–267, 1989.

[64] G. Ciardo, J. Muppala, and K. Trivedi, “SPNP: Stochastic Petri net package,” in *Proc. Third Int. Workshop on Petri Nets and Performance Models*, 1989, pp. 142–151.

[65] W. H. Sanders, W. D. Obal, M. A. Qureshi, and F. K. Widjanarko, “The UltraSAN modeling environment,” *Performance Evaluation*, vol. 24, no. 1, pp. 89–115, October-November 1995.

[66] D. D. Deavours, G. Clark, T. Courtney, D. Daly, S. Derisavi, J. M. Doyle, W. H. Sanders, and P. G. Webster, “The Möbius framework and its implementation,” *IEEE Trans. on Soft. Eng.*, vol. 28, no. 10, pp. 956–969, October 2002.

[67] K. Vaidyanathan and K. Trivedi, “A measurement-based model for estimation of resource exhaustion in operational software systems,” in *Proc. of the Tenth Int. Symp. on Soft. Rel. Engineering*, Nov. 1999, pp. 84–93.

[68] M. C. Hsueh, R. Iyer, and K. Trivedi, “Performability modeling based on real data: A case study,” *IEEE Transactions on Computers*, vol. C-37, no. 4, pp. 478–484, Apr. 1988.

[69] D. Chen, D. Selvamuthu, D. Chen, L. Li, R. R. Some, A. P. Nikora, and K. Trivedi, “Reliability and availability analysis for the JPL remote exploration and experimentation system,” in *Proc. Int. Conf. on Dependable Systems and Networks*, June 2002, pp. 337–344.

[70] J. K. Muppala, A. S. Sathaye, R. C. Howe, and K. S. Trivedi, “Dependability modeling of a heterogenous VAXcluster system using stochastic reward nets,” in *Hardware and Software Fault Tolerance in Parallel Computing Systems*. Ellis Horwood Ltd., 1992, pp. 33–59.

[71] V. Mainkar and K. Trivedi, “Sufficient conditions for existence of a fixed point in stochastic reward net-based iterative models,” *IEEE Trans. on Soft. Eng.*, vol. 22, no. 9, Sept. 1996.

[72] L. Tomek and K. Trivedi, “Fixed-point iteration in availability modeling,” in *Informatik-Fachberichte, Vol. 283: Fehlertolerierende Rechensysteme*. Springer-Verlag, Berlin, 1991, pp. 229–240.

[73] J. G. Kemeney and J. L. Snell, *Finite Markov Chains*. D. Van Nostrand Company, Inc., 1960.

[74] P. Buchholz, “Exact and ordinary lumpability in finite Markov chains,” *Journal of Applied Probability*, vol. 31, pp. 59–74, 1994.

[75] ——, “Efficient computation of equivalent and reduced representations for stochastic automata,” *Int. Journal of Computer Systems Science & Engineering*, vol. 15, no. 2, pp. 93–103, 2000.

[76] R. Milner, *Communication and Concurrency*. London: Prentice Hall, 1989.

[77] P. C. Kanellakis and S. A. Smolka, “CCS expressions, finite state processes, and three problems of equivalence,” in *Proc. ACM Symposium on Principles of Distributed Computing*, 1983, pp. 228–240.

[78] R. Paige and R. E. Tarjan, “Three partition refinement algorithms,” *SIAM J. Comput.*, vol. 16, no. 6, pp. 973–989, 1987.

[79] J. C. Fernandez, “An implementation of an efficient algorithm for bisimulation equivalence,” *Sci. Comput. Programming*, vol. 13, no. 2-3, pp. 219–236, 1990.

[80] M. Bernardo and R. Gorrieri, “A tutorial on EMPA: A theory of concurrent processes with nondeterminism, priorities, probabilities and time,” *Theoretical Computer Science*, vol. 202, pp. 1–54, 1998.

[81] D. T. Huynh and L. Tian, “On some equivalence relations for probabilistic processes,” *Fundamenta Informaticae*, vol. 17, pp. 211–234, 1992.

[82] S. Derisavi, H. Hermanns, and W. H. Sanders, “Optimal state-space lumping in Markov chains,” *Information Processing Letters*, vol. 87, no. 6, pp. 309–315, September 2003.

[83] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad, “Stochastic well-formed colored nets and symmetric modeling applications,” *IEEE Trans. on Computers*, vol. 42, no. 11, pp. 1343–1360, November 1993.

[84] W. H. Sanders and J. F. Meyer, “Reduced base model construction methods for stochastic activity networks,” *IEEE Journal on Selected Areas in Communications*, vol. 9, no. 1, pp. 25–36, Jan. 1991.

[85] W. D. Obal II, “Measure-adaptive state-space construction methods,” Ph.D. dissertation, University of Arizona, 1998.

[86] H. Hermanns and M. Ribaudo, “Exploiting symmetries in stochastic process algebras,” in *Proc. of 12th European Simulation Multiconference (ESM)*, 1998, pp. 763–770.

[87] S. Gilmore, J. Hillston, and M. Ribaudo, “An efficient algorithm for aggregating PEPA models,” *IEEE Transactions on Software Engineering*, vol. 27, no. 5, pp. 449–464, May 2001.

[88] P. Buchholz, “Exact performance equivalence: An equivalence relation for stochastic automata,” *Theoretical Computer Science*, vol. 215, no. 1/2, pp. 263–287, 1999.

[89] ——, “Hierarchical Markovian models: Symmetries and reduction,” *Performance Evaluation*, vol. 22, no. 1, pp. 93–110, February 1995.

[90] ——, “Markovian process algebra: Composition and equivalence,” in *Proc. 2nd Workshop on Process Algebras and Performance Modelling*, ser. Arbeitsberichte des IMMD, vol. 27, no. 4, 1994, pp. 11–30.

[91] ——, “Equivalence relations for stochastic automata networks,” in *Computation with Markov Chains*, W. J. Stewart, Ed. Kluwer Int. Publishers, 1995, pp. 197–216.

[92] ——, “A framework for the hierarchical analysis of discrete event dynamic systems (habilitations thesis),” Ph.D. dissertation, Universität Dortmund, Germany, 1996.

[93] H. Hermanns, *Interactive Markov Chains and the Quest for Quantified Quality*, ser. LCNS. Springer, 2002, vol. 2428.

[94] P.-J. Courtois and P. Semal, “Computable bounds for conditional steady-state probabilities in large Markov chains and queueing models,” *IEEE Journal on Selected Areas in Communications*, vol. SAC-4, no. 6, pp. 926–937, September 1986.

[95] P. J. Courtois, *Decomposability*. New York: Academic Press, 1977.

[96] A. Bobbio and K. Trivedi, “An aggregation technique for the transient analysis of stiff Markov chains,” *IEEE Transactions on Computers*, vol. C-35, no. 9, pp. 803–814, Sept. 1986.

[97] A. Bobbio and K. S. Trivedi, “Computing cumulative measures of stiff Markov chains using aggregation,” *IEEE Transactions on Computers*, vol. 39, no. 10, pp. 1291–1297, 1990.

[98] D. Daly, P. Buchholz, and W. H. Sanders, “An approach for bounding reward measures in Markov models using aggregation,” University of Illinois at Urbana-Champaign Coordinated Science Laboratory, Technical Report UILU-ENG-04-2206 (CRHC-04-06), July 2004.

[99] A. Srinivasan, T. Kam, S. Malik, and R. E. Brayton, “Algorithms for discrete function manipulation,” in *Proc. of the Int’l Conf. on CAD (ICCAD’90)*, 1990, pp. 92–95.

[100] E. M. Clarke, O. Grumberg, and D. Peled, *Model Checking*. MIT Press, 1999.

[101] G. Ciardo, G. Lüttgen, and R. Siminiceanu, “Saturation: An efficient iteration strategy for symbolic state-space generation,” in *Proc. of TACAS 2001*, ser. LNCS, vol. 2031. Springer, 2001, pp. 328–342.

[102] G. Ciardo, R. M. Marmorstein, and R. Siminiceanu, “Saturation unbound,” in *Proc. of TACAS 2003*, ser. LNCS, vol. 2619. Springer, 2003, pp. 379–393.

[103] B. Plateau, “On the stochastic structure of parallelism and synchronization models for distributed algorithms,” in *Proc. of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, Austin, Texas, United States, 1985, pp. 147–154.

[104] B. Plateau and K. Atif, “Stochastic automata network for modeling parallel systems,” *IEEE Trans. in Soft. Eng.*, vol. 17, no. 10, pp. 1093–1108, Oct. 1991.

[105] P. Buchholz, “Numerical solution methods based on structured descriptions of Markovian models,” in *Computer Performance Evaluation*. Elsevier Science Publishers B.V. (North-Holland), 1991, pp. 251–267.
[106] P. Buchholz and P. Kemper, “Numerical analysis of stochastic marked graphs,” in *Proc. 6th Int. Workshop on Petri Nets and Performance Models (PNPM’95)*, October 1995, pp. 32–41.
[107] S. Donatelli, “Superposed stochastic automata: A class of stochastic Petri nets amenable to parallel solution,” in *Proc. 4th Int. Workshop on Petri Nets and Performance Models*, 1991, pp. 54–63.
[108] ——, “Superposed generalized stochastic Petri nets: Definition and efficient solution,” in *Proc. 15th Int. Conf. on Applications and Theory of Petri Nets*, ser. LNCS, vol. 815. Springer-Verlag, 1994, pp. 258–277.
[109] P. Kemper, “Numerical analysis of superposed GSPNs,” in *Proc. 6th Int. Workshop on Petri Nets and Performance Models (PNPM’95)*, 1995, pp. 52–61.
[110] P. Buchholz, G. Ciardo, S. Donatelli, and P. Kemper, “Complexity of memory-efficient Kronecker operations with applications to the solution of Markov models,” *INFORMS J. on Computing*, vol. 12, no. 3, pp. 203–222, 2000.
[111] G. Ciardo and A. Miner, “A data structure for the efficient Kronecker solution of GSPNs,” in *Proc. 8th Int. Workshop Petri Nets and Performance Models*, 1999, pp. 22–31.
[112] A. S. Miner, “Efficient solution of GSPNs using canonical matrix diagrams,” in *Proc. 9th International Workshop on Petri Nets and Performance Models*, Aachen, Germany, September 2001, pp. 101–110.
[113] D. D. Deavours and W. H. Sanders, “An efficient disk-based tool for solving large Markov models,” *Performance Evaluation*, vol. 33, pp. 67–84, 1998.
[114] ——, “‘On-the-fly’ solution techniques for stochastic Petri nets and extensions,” *IEEE Trans. on Soft. Eng.*, vol. 24, no. 10, pp. 889–902, Oct. 1998.
[115] E. de Souza e Silva and H. R. Gail, “Calculating availability and performability measures of repairable computer systems,” *Journal of the ACM*, vol. 36, pp. 171–193, January 1989.
[116] ——, “Calculating transient distributions of cumulative reward,” in *Proceedings of SIGMETRICS/Performance-95*, Ottawa, Canada, May 1995, pp. 231–240.
[117] 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*, Sendai, Japan, June 1996, pp. 116–125.
[118] V. V. Lam, P. Buchholz, and W. Sanders, “A structured path-based approach for computing transient rewards of large CTMCs,” in *Proceedings of the 1st Int. Conf. on Quantitative Evaluation of Systems (QEST)*, September 2004.
[119] J. Muppala, M. Malhotra, and K. Trivedi, “Stiffness-tolerant methods for transient analysis of stiff Markov chains,” *Microelectronics and Reliability*, vol. 34, no. 11, pp. 1825–1841, 1994.
[120] A. Reibman and K. S. Trivedi, “Numerical transient analysis of Markov models,” *Computers and Operations Research*, vol. 15, no. 1, pp. 19–36, 1988.
[121] A. van Moorsel and W. H. Sanders, “Adaptive uniformization,” *ORSA Communications in Statistics: Stochastic Models*, vol. 10, no. 3, pp. 619–648, August 1994.
[122] ——, “Transient solution of markov models by combining adaptive & standard uniformization,” *IEEE Transactions on Reliability*, vol. 46, no. 3, pp. 430–440, September 1997.
[123] M. Malhotra and A. Reibman, “Selecting and implementing phase approximations for semi-Markov models,” *Commun. Statist. Stochastic Models*, vol. 9, no. 4, pp. 473–506, 1993.
[124] S. Gokhale and K. Trivedi, “A time/structure based software reliability model,” *Annals of Software Engineering*, vol. 8, pp. 85–121, 1999.
[125] S. Gokhale, P. N. Marinos, M. R. Lyu, and K. Trivedi, “Effect of repair policies on software reliability,” in *Proc. 12th Annual Conference on of Computer Assurance (COMPASS)*, June 1997, pp. 105–116.
[126] R. Geist, M. Smotherman, K. S. Trivedi, and J. B. Dugan, “Reliability analysis of life-critical systems,” *Acta Informatica*, vol. 23, no. 6.
[127] G. Ciardo, R. A. Marie, B. Sericola, and K. S. Trivedi, “Performability analysis using semi-Markov reward processes,” *IEEE Trans. Computers*, vol. 39, no. 10, pp. 1251–1264, Oct. 1990.
[128] V. Kulkarni, *Modeling and Analysis of Stochastic Systems*. New York: Chapman Hall, 1995.
[129] M. Ajmone Marsan, G. Balbo, and G. Conte, “A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems,” *ACM Transactions on Computer Systems*, vol. 2, no. 2, pp. 93–122, 1984.
[130] J. F. Meyer, A. Movaghar, and W. H. Sanders, “Stochastic activity networks: Structure, behavior, and application,” in *Proc. International Workshop on Timed Petri Nets*, Torino, Italy, July 1985, pp. 106–115.
[131] W. H. Sanders and J. F. Meyer, “Stochastic activity networks: Formal definitions and concepts,” in *Lectures on Formal Methods and Performance Analysis, First EEF/Euro Summer School on Trends in Computer Science*, ser. LNCS, no. 2090, 2001, pp. 315–343.
[132] J. Hillston, *A Compositional Approach to Performance Modelling*. Cambridge University Press, 1996.
[133] H. Hermanns and M. Rettelbach, “Syntax, semantics, equivalences, and axioms for MTIPP,” in *Proc. 2nd Workshop on Process Algebras and Performance Modelling*, ser. Arbeitsberichte des IMMD, vol. 27, no. 4, 1994, pp. 71–87.
[134] M. Malhotra and K. Trivedi, “Dependability modeling using Petri nets,” *IEEE Transactions on Reliability*, vol. 44, no. 3, pp. 428–440, Sept. 1995.
[135] J. B. Dugan, V. Nicola, R. Geist, and K. Trivedi, “Extended stochastic Petri nets: Applications and analysis,” in *Performance’84*, 1985, pp. 507–519.
[136] M. A. Marsan and G. Chiola, “On Petri nets with deterministic and exponentially distributed firing times,” in *Advances in Petri Nets 1987*, ser. LNCS, vol. 266. Springer, 1987, pp. 132–145.
[137] H. Choi, V. Kulkarni, and K. Trivedi, “Markov regenerative stochastic Petri nets,” *Performance Evaluation*, vol. 20, pp. 337–357, 1994.
[138] V. Catania, A. Puliafito, M. Scarpa, and L. Vita, “Concurrent generalized Petri nets,” *Proc. of Numerical Solution of Markov Chains*, pp. 359–382, January 1995.
[139] G. Horton, V. Kulkarni, D. Nicol, and K. S. Trivedi, “Fluid stochastic Petri nets: Theory, application, and solution techniques,” *European Journal of Operations Research*, vol. 105, no. 1, pp. 184–201, Feb. 1998.
[140] G. Ciardo, D. M. Nicol, and K. S. Trivedi, “Discrete-event simulation of fluid stochastic Petri nets,” *IEEE Transactions on Software Engineering*, vol. 25, no. 2, pp. 207–217, 1999.
[141] R. Ortalo, Y. Deswarte, and M. Kaâniche, “Experimenting with quantitative evaluation tools for monitoring operational security,” *IEEE Transactions on Software Engineering*, vol. 25, pp. 633–650, Oct. 1999.
[142] B. Littlewood, S. Brocklehurst, N. Fenton, P. Mellor, S. Page, and D. Wright, “Towards operational measures of computer security,” *Journal of Computer Security*, vol. 2, pp. 211–229, 1993.
[143] E. Jonsson and T. Olovsson, “A quantitative model of the security intrusion process based on attacker behavior,” *IEEE Transactions on Software Engineering*, vol. 23, no. 4, pp. 235–245, April 1997.
[144] M. Dacier, Y. Deswarte, and M. Kaâniche, “Quantitative assessment of operational security: Models and tools,” LAAS, Tech. Rep. 96493, May, 1996.
[145] D. Wang, B. Madan, and K. Trivedi, “Security analysis of SITAR intrusion-tolerant system,” in *Proc. ACM Workshop on Survivable and Self-Regenerative Systems*, 2003, pp. 23–32.
[146] W. H. Sanders, M. Cukier, F. Webber, P. Pal, and R. Watro, “Probabilistic validation of intrusion tolerance,” in *Supplemental Volume Int. Conference on Dependable Systems & Networks (DSN-2002)*, Washington, DC, June 2002, pp. B–78–B–79.
[147] http://www.nessus.org/.
[148] http://www.insecure.org/nmap/.
[149] D. Farmer and E. H. Spafford, “The COPS security checker system,” in *Proc. Summer Usenix Conference*, Berkeley, CA, 1990, pp. 165–170.
[150] http://www.net.tamu.edu/network/tools/tiger.html.
[151] A. Sharma, J. R. Martin, N. Anand, M. Cukier, and W. H. Sanders, “Ferret: A host vulnerability checking tool,” in *Proc. 10th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC-10)*, Papeete, Tahiti, French Polynesia, March 2004, pp. 389–394.
[152] P. Heidelberger, “Fast simulation of rare events in queueing and reliability models,” *ACM Transactions on Modeling and Computer Simulation*, vol. 1, no. 5, pp. 43–85, 1995.
[153] J. Banks, J. Carson, B. Nelson, and D. Nicol, *Discrete-Event System Simulation*. Upper Saddle River, NJ: Prentice-Hall, 2000.
[154] G. Shedler, *Regenerative Stochastic Simulation*. Boston: Prentice-Hall, 1993.
[155] R. E. Nance, “A history of discrete event simulation programming languages,” in *Proc. 2nd ACM SIGPLAN Conference on History of Programming Languages*. ACM Press, 1993, pp. 149–175.
[156] http://www.isi.edu/nsnam/ns/.
[157] http://www.ssfnet.org.

[158] D. Moore, C. Shannon, and K. Claffy, "Code-red: A case study on the spread and victims of an Internet worm," in *Proc. of the Internet Measurement Workshop (IMW)*, November 2002, pp. 273–284.

[159] D. Nicol, M. Liljenstam, and J. Liu, "Multiscale modeling and simulation of worm effects on the Internet routing infrastructure," in *Proceedings of 13th International Conference on Modeling Techniques and Tools for Computer Performance Evaluation (Performance TOOLS 2003)*, Urbana, IL, September 2003.

[160] M. Liljenstam, Y. Yuan, B. Premore, and D. Nicol, "A mixed abstraction level model of large-scale Internet worm infestations," in *Proceedings of the Tenth IEEE/ACM Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, October 2002, pp. 109–116.

[161] C. C. Zou, L. Gao, W. Gong, and D. Towsley, "Monitoring and early warning for Internet worms," in *Proceedings of the 10th ACM Conference on Computer and Communication Security*, 2003, pp. 190–199.

[162] C. C. Zou, W. Gong, and D. Towsley, "Code red worm propagation modeling and analysis," in *Proceedings of the 9th ACM Conference on Computer and Communications Security*, 2002, pp. 138–147.

[163] V. Venkataraghavan, S. Nair, and P.-M. Seidel, "Simulation-based validation of security protocols," in *Proceedings of OPNETWORKS 2002*, August 2002.

[164] D. Apostal, T. Foote-Lennox, T. Markham, A. Dowd, R. Lu, and D. O'Brian, "Checkmate network security modeling," in *Proceedings of the DARPA Information Survivability Conference and Exposition*, June 2001, pp. 214–226.

[165] V. Gorodetski, I. Kotenko, and O. Karsaev, "Multi-agent technologies for computer network security: Attack simulation, intrusion detection and intrusion detection learning," *International Journal of Computer Systems Science and Engineering*, vol. 18, no. 4, pp. 191–200, July 2003.

[166] N. Falby, M. Thompson, and C. Irvine, "A security simulation game definition language," in *Innovative Program Abstracts - Colloquium on Information Systems Security Education*, West Point, NY, June 2004.

[167] C. Irvine and M. Thompson, "Teaching objectives of a simulation game for computer security," in *Proceedings of Informing Science and Information Technology Joint Conference*, June 2003.

[168] J. Drew, "Simulation to support security issues related to system interoperability," in *Proc. of Summer Simulation Conference*, 2002, pp. 14–18.

[169] S. Lathrop, J. Hill, and J. Surdu, "Modeling network attacks," in *Proceedings of the 12th Conference on Behavior Representation in Modeling and Simulation*, May 2003, pp. 401–407.

[170] W. Dizard III, "Seattle cybergame preceded last week's drill and simulated reality," *Government Computer News*, vol. 22, no. 11, 2003, http://www.gcn.com/22_11/news/22099-1.html.

[171] T. Bridis, "Gov't simulates terrorist cyberattack," in Associated Press, November 2003, http://www.zone-h.org/en/news/read/id=3728.

David M. Nicol David M. Nicol is Professor of Electrical and Computer Engineering at the University of Illinois at Urbana-Champaign. He is a long-time contributor in the field of parallel and distributed discrete-event simulations, having written one of the early Ph.D. theses on the topic. He has also worked in parallel algorithms, algorithms for mapping workload in parallel architectures, performance analysis, and reliability modeling and analysis. His research contributions extend to approximately 150 articles in leading computer science journals and conferences, and he is co-author of the textbook *Discrete-Event Systems Simulation*. His current interests lie in modeling and simulation of very large systems, particularly communications and other infrastructure, with applications in security. He was co-architect of the *Scalable Simulation Framework* (SSF) and several of its implementations, now in wide use for network analysis in education, industry, and government. From 1997 to 2003 he was the Editor-in-Chief of the ACM Transactions on Modeling and Computer Simulation, and from 2002-2003 he served as the Associate Director for Research and then the Acting Director of the Institute for Security Technology Studies, at Dartmouth College. Professor Nicol is a Fellow of the IEEE.

William H. Sanders William H. Sanders is a Professor in the Department of Electrical and Computer Engineering and the Coordinated Science Laboratory at the University of Illinois. He is Vice-Chair of IFIP Working Group 10.4 on Dependable Computing. In addition, he serves on the editorial board of *IEEE Transactions on Reliability*, and is the Area Editor for Simulation and Modeling of Computer Systems for the *ACM Transactions on Modeling and Computer Simulation*. He is a past Chair of the IEEE Technical Committee on Fault-Tolerant Computing. He is a Fellow of the IEEE and the ACM. Dr. Sanders's research interests include performance/dependability/security evaluation and dependable and secure computing. He has published approximately 150 technical papers in these areas. He has served as an organizer and on the program committees of numerous conferences and workshops. He is a co-developer of three tools for assessing the performability of systems represented as stochastic activity networks: METASAN, *UltraSAN*, and Möbius. Möbius and *UltraSAN* have been distributed widely to industry and academia; more than 300 licenses for the tools have been issued to universities, companies, and NASA for evaluating the performance, dependability, security, and performability of a variety of systems. He is also a co-developer of the Loki distributed system fault injector and the AQuA/ITUA middlewares for providing dependability/security to distributed and networked applications.

Kishor S. Trivedi Kishor S. Trivedi holds the Hudson Chair in the Department of Electrical and Computer Engineering at Duke University, Durham, NC. He is the Duke-Site Director of an NSF Industry-University Cooperative Research Center between NC State University and Duke University for carrying out applied research in computing and communications. He has been on the Duke faculty since 1975. He is the author of a well-known text entitled *Probability and Statistics with Reliability, Queuing and Computer Science Applications*, with a thoroughly revised second edition being published by John Wiley. He has also published two other books, entitled *Performance and Reliability Analysis of Computer Systems* (published by Kluwer Academic Publishers) and *Queueing Networks and Markov Chains* (John Wiley). His research interests are in reliability and performance assessment of computer and communication systems. He has published over 300 articles and lectured extensively on these topics. He has supervised 37 Ph.D. dissertations. He is a Fellow of the Institute of Electrical and Electronics Engineers. He is a Golden Core Member of the IEEE Computer Society. He is a co-designer of the HARP, SAVE, SHARPE, and SPNP software packages, which have been well-circulated.