THE EVOLUTION OF DEPENDABLE COMPUTING AT THE UNIVERSITY OF ILLINOIS

R. K. Iyer, W. H. Sanders, J. H. Patel, Z. Kalbarczyk
Coordinated Science Laboratory and Electrical and Computer Engineering Dept.
University of Illinois at Urbana-Champaign, USA
{iyer, whs, patel, kalbar}@crhc.uiuc.edu

Abstract: The University of Illinois has been active in research in the dependable computing field for over 50 years. Fundamental ideas have been proposed and major contributions made by researchers at the University of Illinois in the areas of error detection and recovery, fault tolerance middleware, testing and diagnosis, experimental evaluation and benchmarking of system dependability, dependability modeling, and secure system design and validation. This paper traces the origins of these ideas and their development within the University of Illinois, as well as their influence upon research at other institutions, and outlines current research directions.

Key words: error detection and recovery, experimental evaluation of dependability, fault injection, dependability modeling, secure and survivable system design and validation.

1. INTRODUCTION

The University of Illinois, though nestled in cornfields far from any center of industry, has been surprisingly productive in the field of computers. The first electronic digital computer at the University was ORDVAC [92], built for the Ordnance Department and patterned after the machine developed at the Institute for Advanced Study, Princeton. One of the pioneers in the field of fault-tolerant computing was Professor S. Seshu, whose fundamental contributions to fault diagnosis and fault simulation laid the groundwork for continued research in the field. From those beginnings in the late 1950s and early 1960s, the research has continued at a strong pace at the University, where, at present, around 50 faculty and graduate students are active in the area.

Researchers at the University of Illinois have consistently contributed to the dependable computing community. Over the years they actively participated in primary symposiums on dependable systems, including the FTCS International Symposium on Fault-Tolerant Computing (now DSN, the International Conference on Dependable Systems and Networks), both by presentation of papers and by serving as program and general chairs. Program and general chairs who were at the University, or who have come from the University, include Professor Algirdas Avižienis (FTCS-1, 1971), Professor Gernot Metze (FTCS-2, 1972 and FTCS-4, 1974), Professor John Hayes (FTCS-7, 1977), Professor Jacob Abraham (FTCS-11, 1981 and FTCS-19, 1989), Professor Ravi Iyer (FTCS-19, 1989 and FTCS-25, 1995), Professor William H. Sanders (FTCS-29, 1999), and Dr. Zbigniew Kalbarczyk (DSN 2002).

We are delighted with this opportunity to participate in an event honoring Professor Algirdas Avižienis, a distinguished alumnus of the ECE Department of the University of Illinois and a founder of IFIP WG 10.4 and the Fault-Tolerant Computing Symposium.

2. EARLY COMPUTERS

When ILLIAC 1 (essentially a duplicate of ORDVAC) and ILLIAC II were built at the University of Illinois in the 1950s, fault diagnosis consisted of running a battery of programs that exercised different sections of the machine. These test programs typically compared answers computed two different ways (essentially emulating hardware multiplication in software) or tended to stress what was suspected to be a vulnerable part (e.g., punch and subsequently read a continuous stream of characters on paper tape). In ILLIAC I, a vacuum tube computer (about 2,500 tubes, consuming 35 KW), the maintenance engineers found it useful to vary supply and heater voltages by some margins, to tap tubes and chassis with a small plastic mallet while the test programs were running, and to use only replacement tubes that had been aged at least 100 hours. Special tests were used for the electrostatic *Williams-tube* memory to determine the *Read-Around Ratio* (RAR) for the day, i.e., the number of times a cell's neighbors could be bombarded between refreshes without altering the cell's contents. An RAR of 300 was considered pretty good [156]. By present-day standards, this approach was very primitive; no attempt was made to model faults systematically or to evaluate precisely which segments of the machine were covered by the tests. Yet the routine, preventive, marginal testing maintenance approach and the clinical experience of the maintenance engineers, coupled with the healthy skepticism of the users who didn't completely trust either their numerical methods or the computer, resulted in a highly reliable operation. Of course, the equivalent of the entire ILLIAC I, including its 1024-word memory, could now be put on one IC chip.

The approach to testing in ILLIAC II, a discrete-component transistor machine put in service in 1961, was quite similar. However, to simplify fault diagnosis, the arithmetic unit's control (involving the equivalent of about 100 flipflops) had been designed to operate asynchronously, using essentially a double handshake for each control signal and its acknowledgement. The basic idea came from the theory of speed-independent circuits developed at the University of Illinois [93]. A large percentage of failures would, therefore, simply cause the control to wait for the next step of the handshake sequence; the missing step could easily be identified from the indicator lights on the flipflops. By contrast, the logic for the lookahead control did not use handshaking and exhibited some failures that were extremely difficult to trace. (Incidentally, a subtle design bug in the arithmetic unit, (-2) * (-2) giving (-4), escaped detection by the tests using pseudo-random operands but was caught after about nine months by a numerical double-check built into a user's program.) Note again that no attempt was made to model faults systematically, although the handshake mechanism used in the ALU control exhibited the basic idea of what is now called *self-checking operation*.

3. TESTING AND DIAGNOSIS

3.1 Fault simulation and automatic test generation

In the early 1960s, S. Seshu [134], [135], [136] developed the *Sequential Analyzer*, which included a set of programs that can generate fault simulation data (for single, logical, stuck-line faults) for a given logic circuit and a given test sequence and also has the ability to generate test sequences for combinational as well as sequential circuits. Although these test sequences usually were not minimal, they were generated automatically. The Sequential Analyzer was applied directly, but on a limited scale, at Bell Telephone Laboratories to check for design errors in IC designs prior to production, to generate test sequences that could be incorporated into factory test equipment, and to improve diagnosis procedures for the No. 1 Electronic Switching System (ESS-1). It was also used extensively at the University of Illinois to study computer self-diagnosis when an unduplicated processor was performing checkout and diagnosis of itself [88], [90]. Again, this self-diagnosis procedure relied on the idea that a processor fault could cause the processor to stop prematurely. Fault simulators were soon available commercially. Chang, Manning, and Metze produced, as a tribute to Seshu, what is probably the first book devoted entirely to digital fault diagnosis

[20]. Marlett went on to write the first successful commercial software for automatic test generation in sequential circuits [89].

In the 1980s research continued on fault simulation and automatic test pattern generation (ATPG). Cheng completed his Ph.D. at Illinois under Patel and went on to produce the sequential circuit ATPG at AT&T Bell Labs [25]. Cheng later launched a start-up company (Check Logic Inc.) to produce commercial ATPG and fault simulation tools. Check Logic was later acquired by Mentor Graphics. Cheng's ATPG tools are still being offered by Mentor Graphics. Niermann, also a student of Patel, advanced the APTG and fault simulation algorithms even further [98]. Niermann and Patel launched Sunrise Test Systems in 1989 to productize the research tools from Illinois. Through subsequent acquisitions, Sunrise became a part of Synopsys Inc., which continues to offer these test tools and their derivatives. One of the most notable breakthroughs that came out of this research was a very fast, memory-efficient fault simulation algorithm, PROOFS [99]. The then-entrenched algorithm, Concurrent Fault Simulation [151], was replaced by PROOFS throughout academia and industry and is still the algorithm of choice for fault simulation of synchronous sequential circuits for simulating stuck-at faults and many other fault types. A number of researchers have used the PROOFS fault simulator for research in fault diagnosis, transient error propagation, and logic verification.

As the IC chip size grew, interconnect faults, such as opens and shorts, became far more dominant than transistor faults in the semiconductor manufacturing process. PROOFS was readily adapted to simulate resistive shorts in ICs [52], [114]. With the availability of accurate resistive bridge fault simulators, an ATPG for such faults was not far behind [37].

The late 1990s and beyond brought to the forefront the problem of test application time and test data volume in large scan-based circuits with close to a million flip-flops in scan. Research at Illinois on generating the smallest set of test vectors produced theoretical lower bounds as well as the vector set meeting these bounds in a majority of benchmark circuits [57]. However, reduction of vectors alone was not enough to bring down the test application time and data by large factors. Large reduction was achieved with a novel scan organization called the Illinois Scan Architecture [58], [61]. Illinois Scan is now in use in many large chips.

3.2 Fault models

Another area in which the research done at the University of Illinois proved to be seminal is fault representation [130], [131], [132]. Indistinguishable or dominated faults can easily be identified, a priori, from the network structure and be eliminated from further consideration. Other research in fault representation was carried out, primarily at Stanford University [91] and at Western Electric Company [148]. Further work by Hayes, Smith, and Metze at the University of Illinois concentrated on the analysis of multiple faults, including masking relationships [16], [60], and some surprising results concerning the undetectability of certain multiple faults, i.e., multiple redundancies that have no sub-redundancies [140] and extensions [138].

Detailed studies (by Banerjee and Abraham) at the transistor level indicated that the conventional stuck-at fault model is inadequate for modeling the effects of physical failures on MOS circuits [9]. Those studies were used to develop accurate, higher-level fault models for modules such as decoders and multiplexers, which include the effects of realistic physical failures. A new logical model, in the form of a multivalued algebra, has also been developed [11]. It can be used to model the effects of physical failures at the transistor level, since the model allows for strong interactions among all three terminals of a transistor. As interconnect became dominant in today's large multilayer chips, the focus shifted from transistor defects to interconnect defects, specifically bridges between signal lines in metal. As mentioned in the previous section, a number of fault simulation and ATPG tools were developed to target bridge faults [37], [52], [114].

3.3 Functional-level test generation

As complex chips such as memories and microprocessors began to be used widely in systems because of their increasing density and decreasing cost, the problem of testing these chips without the availability of information about their internal structure became acute. An interesting solution to this problem was initially obtained in the case of memories, for which a higher, functional-level fault model was developed; it was used as the basis for deriving tests. Thus, the initial fault model for memories included stuck bits in the memory as well as coupling between cells in a memory. An $O(n\log_2 n)$ algorithm, which will detect all the faults in the fault model, was developed by Thatte and Abraham [146]. This test generation algorithm was improved by Nair, Thatte, and Abraham [94] to one of complexity $O(n)$. The work was extended by others, including Suk and Reddy [145] at Iowa.

An extrapolation of the approach to testing memories, using only functional-level information, was used by Thatte and Abraham [147] to develop test generation procedures for microprocessors in a user environment. A general graph-theoretic model was developed at the register-transfer level to model any microprocessor using only information about its instruction set and the functions performed. A fault model was developed on a functional level, quite independent of the implementation details. These were used to generate test patterns for microprocessors. A fault simulation study on a real microprocessor showed extremely good fault coverage for tests developed using these procedures.

3.4 Testable design of regular structures

Techniques for deriving testable structures from high-level descriptions were studied in [1]. The generated structures in that case were cellular and interconnected in a tree structure, and a general algorithm to test those tree structures that grows only linearly with the size of the tree was developed.

In 1985, Cheng and Patel [27], [28], [29] developed a comprehensive theory of testing for multiple failures in iterative logic arrays. The theory provided the necessary and sufficient conditions for deriving small test sets and showed that testing for multiple faults required only slightly more effort than testing for single faults. Techniques for testing VLSI bit-serial processors and designing them for testability were also studied in [39].

3.5 Diagnosis and repair

The basic idea of computer self-diagnosis, posed (by Metze) in simplified form as a problem on the Electrical Engineering Ph.D. qualifying examination given in December 1965, led to abstract questions of mutual diagnosis of several computers; those questions were collectively known as the Connection Assignment Problem [109]. That study created an enormous amount of interest, and systems that use different fault models, more general test outcomes, other measures of diagnosability, probabilistic fault diagnosis, and diagnosis of intermittent faults are still being investigated at several different institutions. A survey paper [50] lists 26 derived papers. The idea of not having a global supervisor that detects failures and removes failed units was investigated in [3], in which a new technique for distributed systems, called *roving diagnosis*, was presented.

Diagnosis and repair became important in improving the yield of rectangular arrays of logic with spare rows and columns. Such arrays are common in processor arrays, PLAs, and static and dynamic random access memory chips (RAMs). Fuchs developed a graph theoretic model of such arrays to represent the relationship between faults and spares [78]. The model was used to improve the yield of RAMs [21]. Following that work, many others have published work on RAM yield improvement. In addition to RAMs, these methods are also in use for large cache memories in present-day microprocessors.

Logic diagnosis of defective chips became important in the 1990s as the chips began to have multi-millions of logic gates. Traditional methods of using full fault dictionaries were running into trouble from the explosive growth in the size of the dictionaries. The problem of size was first addressed at Illinois by Fuchs in collaboration with Intel Corp. [119]. Fuchs continued his work on diagnosis with many of his Ph.D. students [14], [59]. Much of that work is used today by semiconductor manufacturers for failure analysis of defective chips.

4. ERROR DETECTION AND RECOVERY

4.1 Self-checking circuits

Self-diagnosis concepts, coupled with the earlier, fundamental results on "dynamically checked computers" by Carter and Schneider [15], led to the formulation of Totally Self-Checking (TSC) circuits by Anderson and Metze [5], [6]. A TSC circuit uses inputs and outputs that are encoded in a suitable code, together with a TSC checker that indicates whether the output is a code word or a non-code word. TSC circuits satisfy the following properties: (1) only code word inputs are needed to diagram the circuit completely (self-checking property), and (2) no fault causes the circuit to output an incorrect code word, i.e., the output is either the correct code word or is an incorrect, non-code word (fault-secure property). Actually, these requirements can be relaxed somewhat for strongly fault-secure networks, which are a larger class of networks that achieve the totally self-checking goal [139], [141]. The main advantages of TSC circuits are that transient errors are either caught or have no effect, that the outputs can be trusted as long as the checkers indicate no error (i.e., that erroneous information is not propagated), and that the circuit diagnoses itself with normally occurring inputs. However, since the normally occurring inputs do not necessarily cycle through the inputs required for a complete test, the circuit nevertheless has to be taken out of service periodically for testing. It should also be mentioned that the problem of finding a code whose error protection capabilities match the error-generation capabilities of the logic is non-trivial (e.g., [48]). TSC research has also led to numerous extensions at other institutions, including Bell Telephone Laboratories, Stanford University, USC, and the University of Iowa, among others. The strongly fault-secure concept has also been adapted and extended, for example, by Jansch and Courtois [68].

4.2 Time redundancy

The use of time redundancy for checking errors in hardware gained renewed attention in a series of papers from Illinois starting in the late 1970s. The papers dealt with a variety of techniques and circuits. The first in a series of these results was a report on the fault detection capabilities of alternating logic in circuits by Reynolds and Metze [118]. The alternating logic used circuits, which were arranged to be functionally self-dual. Conditions were presented that, if satisfied by a circuit, guaranteed the detection of errors due to single stuck-at faults. The paper also discussed the application of alternating logic and sequential logic.

A method of error detection called *Recomputing with Shifted Operands (RESO)* was proposed and analyzed for arithmetic and logic units (ALU) by Patel and Fung [105]. That was the first time that a unified method was used for both arithmetic and logic operations. That paper also differed from the earlier papers on self-checking logic in that it assumed a far more general fault model, which was suitable for the emerging VLSI circuits. Depending on the number of shifts used for the recomputed step, a variable amount of fault coverage was provided. For example, if *k* shifts were used in an adder, then any (*k - 1*) consecutive failed cells would be covered, although the cells may fail in any arbitrary way. The method of RESO was then applied to more complex circuits of multiply and divide arrays [106]. The method was extended to arbitrary one-dimensional iterative logic arrays [26]. Variants of RESO were also used for error correction in arithmetic operations [79] and data transmission [108].

4.3 Memory error detection and recovery

Abraham, Davidson, and Patel developed a new memory system design for tolerating errors due to single-event radiation upsets [2]. The design used coding, control duplication, and scrubbing to tolerate soft errors from single-event upsets, and had much lower cost than a straightforward application of redundancy. An analytical model for reliability of memory with scrubbing was also developed, and is now widely used in industry [121].

4.4 Application-aware techniques

The field of ***control-flow checking*** has been the focus of intense research over the last two decades and resulted in a number of hardware- and/or software-based schemes. Most of the existing solutions, however, are not preemptive in nature, i.e., often the system crashes before any error detection is triggered. PECOS (Preemptive Control Signatures) techniques developed by Bagchi and Iyer enable preemptive detection of errors in the execution flow of an application [7], [8]. The strengths of the technique are that (1) it is preemptive, i.e., the detection happens before a branch/jump is taken, and (2) it significantly reduces events of silent data corruption. PECOS was applied and evaluated on a call-processing application. Fault/error injection results show that use of PECOS eliminates silent data corruptions and application hangs, and the crash incidences of the entire call-processing application are reduced by almost three times. A generalization of preemptive checking is instruction checking, which is a focus of current work [95].

Data audits are traditionally used in the telecommunications industry and implement a broad range of custom and ad hoc application-level techniques for detecting and recovering from errors in a switching environment. In [8] Liu, Kalbarczyk, and Iyer presented design, implementation, and assessment of a dependability framework for a call-processing environment in a digital mobile telephone network controller. The framework contains a data audit subsystem to maintain the structural and semantic integrity of the database. The fault-injection-based evaluation of the proposed solution indicates that the data audit detects 85% of the errors and significantly reduces the incidence of escaped errors.

The semantic of application programs usually exhibits instruction-level parallelism, which can be exploited in a super-scalar architecture for increasing performance. However, there is a limit to this parallelism due to dependency between instructions, which forces the processor to execute the dependent instructions in separate cycles. For example, one instruction may use the result produced by a previous instruction as its operand. Failure of the processor to execute the instructions in the correct sequence may adversely affect the output of the program and therefore should be considered an error. ***Instruction sequence checking*** verifies (through monitoring the issue and execution of the instructions in the pipeline at runtime) whether a sequence of dependent instructions is executed in the correct order.

4.5 Checkpointing and recovery

Deployment of distributed applications supporting critical services, e.g., banking, aircraft control, or e-commerce, created a need for efficient error recovery mechanisms and algorithms. In this context an important research area, led by Fuchs and his students Wang, Alewine, and Neves, was the development of novel checkpointing and rollback recovery strategies.

Independent checkpointing for parallel and distributed systems allows maximum process autonomy, but suffers from possible domino effects (processes have to roll back an unbounded number of times as they attempt to find a consistent global state for recovery) and storage space overhead for maintaining multiple checkpoints and message logs. In [155] it was shown that transformation and decomposition can be successfully applied to the problem of efficiently identifying all discardable message logs to achieve optimal garbage collection, and hence optimize the space overhead and improve performance of checkpointing schemes.

Another research avenue pursued by Fuchs focused on investigation of the applicability of a compiler-assisted multiple instructions rollback scheme (a technique developed for recovery from transient processor failures) to aid in speculative execution repair. The work took advantage of the fact that many problems encountered during recovery from branch misprediction or from instruction re-execution due to exceptions in speculative execution architecture are similar to those encountered during multiple instructions rollback. Consequently, extensions to the compiler-assisted scheme were added to support branch and exception repair [4].

In subsequent work Neves and Fuchs [96] developed a new low-overhead coordinated checkpoint protocol for long-running parallel applications and high-availability applications. The protocol uses time to avoid all types of direct coordination (e.g., message exchanges and message tagging), reducing the overheads to almost a minimum. To ensure that rapid recoveries can be attained, the protocol guarantees small checkpoint latencies. The protocol was implemented and tested on a cluster of workstations, and the results show very small overhead. In [97] the RENEW toolset for rapid development and testing of checkpoint protocols with standard benchmarks was proposed.

4.6 Algorithm-based fault tolerance

An exciting new direction in the design of fault-tolerant systems was started when Huang and Abraham [63] developed matrix encoding schemes for detecting and correcting errors when matrix operations are performed using processor arrays. The schemes were generalized to the new system-level method of achieving high reliability called *algorithm-based fault tolerance (ABFT)*. The technique encodes data at a high level, and algorithms are designed to operate on the encoded data and produce encoded output data. The computation tasks within the algorithm are appropriately distributed among multiple computation units so that failure of one of the units affects only a portion of the output data, enabling the correct data to be recovered from the encoding [64]. This result was applied to matrix operations using multiple processor arrays. The work was generalized to linear arrays by Jou and Abraham [71] and also extended to Laplace equation solvers [65], as well as FFT networks [72]. [10] developed fault tolerance techniques for three powerful paradigms: the multiplex, the recursive combination, and the multiplex/demultiplex paradigms. In the proposed approach, processors that are idle during normal computation are used to check the results of other processors. In later work, a general theory of algorithm-based fault tolerance was developed that gives bounds on the processor and time overhead in the ABFT scheme [12]. This approach seems to be ideal for low-cost fault tolerance for special-purpose computations, including a wide class of signal-processing applications. The work has been extended in [85]. ABFT has been widely explored by a large number of researchers, more recently as part of the Remote Exploration and Experimentation (REE) program at the Jet Propulsion Laboratory [55].

5. MIDDLEWARE AND HARDWARE SUPPORT FOR FAULT TOLERANCE AND SECURITY

In the 1990s, the high cost of custom hardware solutions, plus the availability of inexpensive COTS hardware, led to development of methods for providing dependability via software middleware. In this spirit, several

University of Illinois projects were initiated that jointly provide reliability and security services. The solution space ranges from purely software approaches (ARMORs [71], AQuA [116], and ITUA [35]) to more recent work on hardware-based (or processor-level) support for error detection, masking of security vulnerabilities, and recovery under one umbrella, in a uniform, low-overhead manner (RSE [95]).

5.1 ARMOR high availability and security infrastructure

The ARMOR approach, proposed by Whisnant, Kalbarczyk, and Iyer, relies on a network of self-checking reconfigurable software modules, which collectively provide high availability and security to applications [71], [158]. The ARMOR infrastructure (consisting of multiple ARMOR processes) is designed to manage redundant resources across interconnected nodes, to foil security threats, to detect errors in both the user applications and the infrastructure components, and to recover quickly from failures when they occur. Because of the flexible ARMOR infrastructure, security protection and detection and recovery services can be added or removed depending on application requirements. The modular design ensures that there is a clear upgrade path through which additional protection capabilities can be added to the ARMOR infrastructure in the future. The architecture has been demonstrated and evaluated (using fault injection) on several real-world applications in the areas of telecommunication (e.g., call processing) [154] and scientific distributed computing (JPL-NAS Mars Rover application) [157].

5.2 A processor-level framework for high dependability and security

A middleware is effective in handling errors as long as they propagate and manifest at that level. Often, however, propagating lower-level errors crash the system (i.e., never reach the middleware level) or cause silent data corruption (i.e., generate latent errors) before being detected. Understanding of that fact, together with an increasing rate of soft errors due to CMOS scaling, led to renewed interest in supporting hardware/processor-based techniques. Recently, Nakka, Kalbarczyk, and Iyer proposed the Reliability and Security Engine (RSE), a hardware-level framework implemented as an integral part of a modern microprocessor and providing application-aware reliability and security support in the form of customizable hardware modules [95]. The detection mechanisms investigated include (1) the Memory Layout Randomization (MLR) Module, which randomizes the memory layout of a process in order to foil attackers who assume a fixed system layout, (2) the Data Dependency Tracking (DDT) Module, which tracks the dependencies among threads of a process and maintains checkpoints of shared memory pages in order to roll back the threads when an offending (potentially malicious) thread is terminated, and (3) the Instruction Checker Module (ICM), which checks an instruction for its validity or the control flow of the program just as the instruction enters the pipeline for execution. Performance simulations for the studied modules indicate low overhead of the proposed solutions.

5.3 AQuA and ITUA

At the network level, three factors have significantly lowered the ability to withstand hostile attacks on critical networked systems: (1) an economic mandate to construct systems with more cost-effective commercial off-the-shelf (COTS) solutions, thereby accepting known and unknown limitations; (2) the increasingly sophisticated nature of commonly available technologies, capable of mounting more complex and sustained attack patterns against these systems; and (3) the fact that systems are increasingly inter-networked and need to remain open to meet interoperability goals. The first of these factors makes it more likely that some systems will be compromised and corrupted by adversaries. The second makes it likely that preplanned, coordinated, and sustained attacks will be mounted against high-value systems. The third implies that effects of successful intrusion will be compounded as multiple systems are impacted. All of these factors have led to recent work in intrusion-tolerant networked systems.

Significant work on designing, implementing, and validating fault- and intrusion-tolerant systems has gone on at the University of Illinois, together with its industrial partners (most notably BBN Technologies) since the late 1990s. In the AQuA project [116], Sanders and his group (including Research Programmer M. Seri) developed the concept of a *property gateway*, which provides adaptation between different types of replication, each providing a different performance/fault-tolerance trade-off, depending on a high-level dependability specification [120],[133]. J. Ren's Ph.D. thesis work [117],[115], completed in 2001, documented much of the AQuA work. In addition to providing differing levels of dependability, AQuA provided tunable consistency and soft real-time performance using algorithms developed by S. Krishnamurthy as part of her Ph.D. thesis [77],[76].

In the ITUA project [35],[104], the AQuA approach was extended to include malicious attacks by combining redundancy management techniques (specifically countering faults resulting from a partially successful attack) and diversity with techniques that produce unpredictable (to the attacker) and variable responses to complicate the ability to preplan a coordinated attack. In the process, new Byzantine algorithms were developed [113],[112] that tolerate the characteristic Byzantine faults resulting from a class of staged, coordinated intrusions.

6. DEPENDABILITY MODELING

6.1 UltraSAN

When Sanders joined the University of Illinois in 1994 from the University of Arizona, he brought with him a team of people working on dependability modeling. Sanders's and his team's work in the early and mid-1990s was implemented in a software tool called *UltraSAN* [129].

UltraSAN was a software package for model-based evaluation of systems represented as stochastic activity networks (SANs) [127]. The model specification process in *UltraSAN* was carried out in a hierarchical fashion. Subsystems, specified as SANs, could be replicated and joined together in a composed model [126] using the "SAN-based reward models" that Sanders had introduced in his Ph.D. thesis in 1988 [122]. On top of the composed model, reward structures could be used to define performance, dependability, and performability measures [128]. To solve a specified model, *UltraSAN* provided analytic solvers [150] (developed by J. Tvedt) as well as discrete-event simulators (developed by R. Friere) [125]. When analytic solvers were used, the state space of the underlying stochastic process was first generated through reduced base model construction [126]. Using that approach, state-space lumping was automatically performed when SANs were replicated in the composed model, thus reducing the state-space size for the models. The models that could be solved analytically by *UltraSAN* included Markov models as well as certain models with deterministic delays [87],[86], which were developed by L. Malhis as part of his Ph.D work. Moreover, an importance sampling component, developed by D. Obal as part of his Master's thesis, was provided to speed up the simulation [101]. *UltraSAN* also contained novel methods for computing the distribution of reward accumulated in a finite interval developed by A. Qureshi as part of his Ph.D thesis [111],[110]. A. van Moorsel developed the theory for, and implemented, adaptive uniformization in *UltraSAN*, which significantly reduces the time to obtain a transient solution of many stiff Markov chains, particularly those that arise in dependability evaluation [152],[153]. Finally, although never implemented in *UltraSAN*, D. Obal significantly extended these methods in his Ph.D. thesis to the more general "graph"-based composed models [100] and path-based reward structures [102],[103].

UltraSAN was licensed a large number of sites for commercial use, and many universities for teaching and research. For example, it was used for many telecommunications applications at Motorola and was the primary dependability evaluation tool in the Iridium project. It has also been used to design disk drive controllers and system-managed storage software at IBM, and ATM and frame-relay networks at US West and Bellcore, among other applications.

6.2 Möbius

By the mid 1990s, it became clear that while the *UltraSAN* approach was successful at evaluating the dependability and performance of many systems, further work was needed to develop performance/dependability modeling frameworks and software environments that could predict the performance of complete networked computing systems, accounting for all system components, including the application itself, the operating system, and the underlying computing and communication hardware.

Ultimately, the experiences with *UltraSAN* showed that a framework should provide a method by which multiple, heterogeneous models can be composed together, each representing a different software or hardware module, component, or aspect of the system. The composition techniques developed should permit models to interact with one another by sharing state, events, or results, and should be scalable. A framework should also support multiple modeling languages (i.e., formalisms), as well as methods to combine models at different levels of resolution. Furthermore, a framework should support multiple model solution methods, including both simulation and analysis, that are efficient. Finally, a framework should be extensible, in the sense that it should be possible to add, with reasonably little effort, new modeling formalisms, composition and connection methods, and model solution techniques.

Those goals were realized in the performance/dependability/security evaluation framework developed at the University of Illinois known as *Möbius* [123],[41]. The first version of Möbius was released in 2001; T. Courtney coordinated the development of this and future versions. The fundamental ideas concerning the Möbius framework were developed by D. Deavours as part of his Ph.D. thesis [43],[42],[40]. Although Möbius was originally developed for studying the reliability, availability, and performance of computer and network systems, its use has expanded rapidly. It is now used for a broad range of discrete-event systems, from biochemical reactions within genes to the effects of malicious attackers on secure computer systems, in addition to the original applications.

That broad range of use is possible because of the flexibility found in Möbius, which comes from its support of multiple high-level modeling formalisms (e.g., Modest [13] and PEPA [34]) and multiple solution techniques. This flexibility allows users to represent their systems in modeling languages appropriate to their problem domains, and then accurately and efficiently solve the systems using the solution techniques best suited to the systems' size and complexity. Time- and space-efficient distributed discrete-event simulation and numerical solution are both supported.

The various components of the Möbius tool are divided into two categories: model specification components and model solution components. The Möbius tool is designed so that new formalisms can be implemented and employed if they adhere to the model-level AFI, as specified in a paper by Deavours and Sanders [43]. The model AFI views models as consisting of two sets of components: state variables, which store model state, and actions, which change model state. This design allows new model formalisms and editors to be incorporated without modification of the existing code, supporting the extensibility of the Möbius tool. Similarly, Derisavi, Kemper, Sanders, and Courtney [46] developed a state-level AFI to cleanly separate numerical solution algorithms from a state-level model representation. A. Christensen developed a connection method by which models could exchange results, in an ordered or fixed-point fashion, to build large system models [32].

Models can be solved, through interface to the state-level AFI, either analytically/numerically or by simulation. Innovative data structures, developed by S. Derisavi as part of his Ph.D work and based on multi-valued decision diagrams, are used to represent system models with tens of millions of states compactly [44],[45]. From each model, C++ source code is generated and compiled, and the object files are linked together to form a library archive [33]. The libraries are linked together along with the Möbius base libraries to form the executable for the solver. Most recently, D. Daly has developed methods for constructing approximate models that have smaller state spaces but bound the error induced by the state space reduction using stochastic ordering arguments [38], and V. Lam has developed path-based methods to solve for instant-of-time variables that do not require explicit representation of either a state-transition-rate matrix or solution vector [80].

Möbius has been distributed widely to other academic and industrial sites. There are now approximately 130 academic and industrial licensees of Möbius, and Illinois is now collaborating with the University of Twente in The Netherlands, Dortmund University, TU Dresden, and the Universität der Bundeswehr München in Germany, and the University of Florence in Italy to further enhance Möbius.

6.3 Depend

In the early 1990s Goswami and Iyer initiated the DEPEND project to develop a framework for designing dependable systems [51]. DEPEND is a simulation-based environment that supports the design of systems for fault tolerance and high availability. It takes as inputs both VHDL and C++ system description and produces as output dependability characteristics including fault coverage, availability, and performance. At the core of DEPEND are simulation engines supported by a fault injector, a set of fault dictionaries, and component libraries. The fault injector provides mechanisms to inject faults. The component libraries contain model-building blocks with detailed functional descriptions and characteristics. The fault dictionaries embody possible fault effects of the given fault types, devices, and circuits. DEPEND was developed with DARPA support and it was licensed to several companies and employed to simulate a number of industrial systems, including Integrity S2 from Tandem (now a division of HP).

7. EXPERIMENTAL EVALUATION / BENCHMARKING OF SYSTEM DEPENDABILITY

7.1 Fault injection

Fault injection has been used since the early days of experimental assessment of dependable systems as a mechanism to evaluate computing systems. At Illinois, research on fault/error injection was driven by failure data analysis of real systems. Tsai and Iyer employed stress-based fault injection to evaluate one of the first UNIX-based fault-tolerant systems developed by Tandem (now a division of HP). The stress-based approach ensures fault/error injection to system components when they are heavily used (i.e., highly stressed) [149]. This allowed meaningful comparison of systems and was an important step towards benchmarking. In order to facilitate automated fault/error injection experiments, NFTAPE, a sophisticated environment for software-implemented automated fault/error injection experiments, was developed [144], [143].

In more recent studies Gu, Kalbarczyk, and Iyer [53], [54] applied error injection to characterize Linux kernel behavior under errors that impact kernel code, kernel data, kernel stack, and processor system registers, and to provide an insight on how processor hardware architecture (instruction set architecture and register set) impacts kernel behavior in the presence of errors. Two target Linux-2.4.22 systems were used: the Intel Pentium 4 (P4) running RedHat Linux 9.0 and the Motorola PowerPC (G4) running YellowDog Linux 3.0. The study found, for example, that (1) the activation of errors is generally similar for both processors, but that the manifestation percentages are about twice as high for the Pentium 4, (2) less-compact fixed 32-bit data and stack access makes the G4 platform less sensitive to errors, and (3) the most severe crashes (those that require a complete reformatting of the file system on the disk) are caused by reversing the condition of a branch instruction. Since the recovery from such failures may take tens of minutes, those failures have a profound impact on availability.

An important research avenue pursued by Xu, Kalbarczyk, and Iyer is the exploration of the possibility of security violations due to errors. In [159] it was shown that naturally occurring hardware errors can cause security vulnerabilities in network applications such as an *FTP* (file transfer protocol) and *SSH* (secure shell). As a result, relatively passive but malicious users can exploit the vulnerabilities. While the likelihood of such events is small, considering the large number of systems operating in the field, the probability of such vulnerabilities cannot be neglected. In the following study, Chen, another student of Iyer, employed fault/error injection to experimentally evaluate and model the error-caused security vulnerabilities and the resulting security violations on two Linux kernel-based firewall facilities (*IPChains* and *Netfilter*) [22]. Using data on field failures, data from the error injection experiments, and system performance parameters such as processor cache miss and replacement rates, a SAN (Stochastic Activity Network) model was developed and simulated to predict the mean time to security vulnerability and the duration of the window of vulnerability under realistic conditions. The results indicate that the error-caused vulnerabilities can be a non-negligible source of security violations.

In parallel with that work, members of Sanders's group, which included R. Chandra, M. Cukier, D. Henke, K. Joshi, R. Lefever, and J. Pistole, worked to develop a new form of fault and attack injection for distributed systems in which the introduction of faults is triggered based on the global state of the system. In addition to developing the basic concepts, and supporting theorems related to global-state-based fault injection (GSBFI), they deployed Loki, a global-state-based fault injector [18],[36],[19],[17], and used it to experimentally evaluate two large-scale distributed systems. In particular, in [70] K. Joshi used Loki to assess the unavailability induced by a group membership protocol in Ensemble, a widely used group communication system. R. Lefever employed Loki to evaluate the effects of correlated network partitions on Coda, a popular distributed file system [82].

There are two benefits of using GSBFI. The first is the ability to validate a system when its fault models rely on states that are hard to target either because they are short-lived or because they occur infrequently. Examples include correlated faults, stress-based faults, and malicious faults. The second benefit is the ability to perform evaluations beyond the scope of fault forecasting. GSBFI can be used to estimate a broad range of conditional measures for use in system models to compute a variety of unconditional performance and dependability measures. Efforts are currently underway to apply GSBFI to the experimental evaluation of the survivability of systems by systematically injecting the effects of cyber attacks in a correlated manner.

7.2 Operational life monitoring and failure data analysis

The use of measured data to study failures in a real use environment has been the focus of active research for quite some time. Rossetti and Iyer [66] used measured data to study the effect of increasing workload on hardware and software fault tolerance. Analysis showed that the probability of a CPU-related error increases nonlinearly with increasing workload. The resulting increase in the error probability can be 50 to 100 times more than that at a low workload. Those results show that reliability models cannot be considered representative unless the system workload environment is taken into account, since the gain in performance is more than offset by degradation in reliability. Similar results relating to operating system reliability appeared in [67]. A novel experiment to obtain, for the first time, distributions of error latency was performed by Chillarege and Iyer [31]. The extension of that work to the study of various fault models appears in [30].

More recent studies by Kalyanakrishnam, Xu, Kalbarczyk, and Iyer focused on error and failure analysis of a LAN of Windows NT-based servers [75], [161] and reliability of Internet hosts [74] with particular focus on the importance of the user's perspective in assessing the systems. For example, while the measured availability of the LAN of Windows-based mail servers is 99%, the user-perceived availability is only 92% [75]. The study on Internet hosts' reliability showed that on average, a host remained unavailable to the user for 6.5 hours (during the 40-day experiment, i.e., approx. 2.5 days per year), which is an availability of about 99%. However, closer analysis of data revealed that an average (or mean value) is not always an adequate measure, because it may hide the reality experienced by the user. For example, a more detailed data breakdown revealed that (1) 45% of hosts had a total downtime ranging from 1,000 seconds to 7,000 seconds, and a median downtime of nearly an hour (i.e, approximately 9.5 hours per year), (2) 49% of hosts had a total downtime ranging from 7,000 seconds to 70,000 seconds and a median downtime of about 4.5 hours (i.e., approximately 40 hours per year), and (3) 6% of hosts had a total downtime ranging from 90,000 seconds to 120,000 seconds, and a median downtime of about 2.2 days (i.e., approx. 20 days per year).

8. SECURE SYSTEM DESIGN AND VALIDATION: FROM DATA ANALYSIS TO PROTECTION AND TOLERANCE MECHANISMS

Challenged by the increasing number and severity of malicious attacks, security has become an issue of primary importance in designing dependable systems. There is no better way to understand the security characteristics of computer systems than by direct measurement and analysis.

8.1 Measurement-driven security vulnerability analysis

In a seminal work, Chen, Kalbarczyk, and Iyer employed a combination of an in-depth analysis of real data on security vulnerabilities and a focused source-code examination to develop a finite state machine (FSM) model to depict and reason about the process of exploiting vulnerabilities and to extract logic predicates that need to be met to ensure vulnerability-free system implementation [24]. In the FSM approach, each predicate is represented as a primitive FSM (*pFSM*), and multiple pFSMs are combined to develop FSM models of vulnerable operations and possible exploitations. The proposed FSM methodology is demonstrated by analysis of several types of vulnerabilities reported in the *Bugtraq* [62] database: stack buffer overflow, integer overflow, heap overflow, file race condition, and format string vulnerabilities, which constitute 22% of all vulnerabilities in the database. For the studied vulnerabilities, three types of pFSMs were identified that can be used to analyze operations involved in exploitation of vulnerabilities and to identify the security checks to be performed at the elementary activity level.

A practical demonstration of the usefulness of the approach was the discovery of a new heap overflow vulnerability now published in *Bugtraq* (ID 6255). The discovery was made during construction of the FSM model for another known vulnerability of the *null HTTPD* application (a multithreaded web server for Linux and Windows platforms).

8.2 Vulnerability avoidance

The low-level analysis of severe security vulnerabilities indicates that a significant number of vulnerabilities are caused by programmers’ improper use of library functions. For example, omitting buffer size checking before calling string manipulation functions, such as *strcpy* and *strcat*, causes many buffer overflow vulnerabilities. A common characteristic of many of these vulnerabilities is pointer taintedness. A pointer is tainted if a user input can directly or indirectly be used as a pointer value. Pointer taintedness that leads to vulnerabilities usually occurs as a consequence of low-level memory writes, typically hidden from the high-level code. Hence, a memory model is necessary for reasoning about pointer taintedness. In [23] the memory model is formally defined and applied to reasoning about pointer taintedness in commonly used library functions.

Reasoning about pointer taintedness makes it possible to extract security preconditions, which either correspond to already known vulnerability scenarios (e.g., format string vulnerability and heap corruption) or indicate the possibility of function invocation scenarios that may expose new vulnerabilities. This work will progress through (1) an investigation of approaches that reduce the amount of human intervention in the theorem-proving tasks using pointer-analysis techniques and heuristics, and (2) the exploration of the possibility of incorporating this technique into compiler-based static checking tools.

8.3 Protection mechanisms against security attacks

FSM-based analysis of vulnerabilities also indicates that security problems, such as buffer overflow, format string, integer overflow, and double-freeing of a heap buffer lead to Unauthorized Control Information Tampering (UCIT) in a target program. An additional survey (other than *Bugtraq*) of the 109 CERT security advisories issued over the past four years shows that UCIT vulnerabilities account for nearly 60% of all the CERT advisories. Transparent Runtime Randomization (TRR), proposed by Xu, Kalbarczyk, and Iyer, is a generalized approach to protect systems against a wide range of security attacks that exploit UCIT vulnerabilities [160]. The TRR technique dynamically and randomly relocates a program's stack, heap, shared libraries, and parts of its runtime control data structures inside the application memory address space. If a program's memory layout is different each time it runs, it foils the attacker's assumptions about the memory layout of the vulnerable program and makes the determination of critical address values difficult if not impossible. An incorrect address value for a critical memory element causes the target application to crash. Although a crash may not be desirable from reliability and availability perspectives, in the security domain, a crash is an acceptable option for the program being hijacked. TRR is implemented by changing the Linux dynamic program loader; hence, it is transparent to applications. TRR incurs less than 9% program startup overhead and no runtime overhead.

8.4 DPASA: Designing protection and adaptation into a survivability architecture

The AQuA and ITUA work, together with other work in intrusion-tolerant systems (e.g., [47],[49]) has suggested that it may be possible to build large-scale, networked intrusion-tolerant systems that can continue to provide specified services even when under sustained partially successful cyber attack. To test this hypothesis, the University of Illinois, together with partners at BBN, SRI, Draper Labs, Adventium Labs, and the University of Maryland, embarked upon a 2-year project called DPASA (Designing Protection and Adaptation into a Survivability Architecture) to design, implement, and validate a large-scale, intrusion-tolerant publish and subscribe system [142]. Team members from Illinois included A. Agbaria, T. Courtney, M. Ihde, J. Meyer (a consultant), W. Sanders, M. Seri, S.

Singh, and F. Stevens. The designed system was considered to be prototypical of many critical communication systems, and a good test of recently developed intrusion-tolerance techniques.

8.5 DPASA architecture

The publish and subscribe system developed consisted of multiple clients communicating with each other through a central core. Redundancy and diversity were used in the core, and the core consisted of four quadrants. Each quadrant was divided into three zones: the crumple zone, the operations zone, and the executive zone. The client-hosted components of the publish-subscribe middleware included a survivability delegate that intercepted the mission application's requests and managed communication with the core, including cryptographic manipulations, through DJM stubs. The crumple zone accommodated client-core communication via multiple access proxies which served as the first barrier between the core and clients after the isolation switch. The operational zone provided the PS&Q functionality and intrusion/fault detection mechanisms, such as the guardians. It contained several components, each with specific tasks. The PSQ component was responsible for performing publish, subscribe, and query operations requested by clients. The guardian and the correlator were responsible for performing intrusion detection in the core and IOs inside it. The downstream controller (DC) and the policy server (PS) components were responsible for specifying policies and forwarding control information to the autonomic distributed firewalls (ADF NICs) [107] installed on the hosts. Finally, the system manager (SM) of the Executive zone managed the core's actions.

The system also included intrusion detection system (IDS) [69] components for improving its survivability. The main components participating in the alert/response data flow were the IDS components: sensors, actuators, and local controllers (LC). IDS components were associated with many of the processing and communication components of the system. Sensors are dedicated to intrusion detection, actuators are mechanisms that carry out actions when commanded, and an LC is a control agent responsible for local survivability management functions.

8.6 DPASA architecture validation

A methodology for validating, in a quantitative manner, the survivability of the DPASA design was also developed. Efforts for quantitative validation of security have usually been based on formal methods [81], or have been informal, using "red teams" to try to compromise a system [84].

Probabilistic modeling has been receiving increasing attention as a mechanism to validate security [83], [124]. For example, work at Illinois by

Singh et al. [137] used probabilistic modeling to validate the ITUA intrusion-tolerant architecture, 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. Shortly thereafter, Gupta et al. [56], in a paper resulting from two class projects in Sanders's graduate class, used a similar approach to evaluate the security and performance of several intrusion-tolerant server architectures. Probabilistic modeling is especially suited to intrusion-tolerant systems, since by definition, intrusion tolerance is a quantitative and probabilistic property of a system.

In the DPASA project, a probabilistic model was used to validate the system design, as documented in F. Stevens's Master's thesis [142]. The probabilistic model made use of an innovative attacker model. The attacker model had a sophisticated and detailed representation of various kinds of effects of intrusions on the behavior of system components (such as a variety of failure modes). It included a representation of the process of discovery of vulnerabilities (both in the operating system(s) and in the specific applications being used by the system) and their subsequent exploitation, and considered an aggressive spread of attacks through the system by taking into account the connectivity of the components of the system at both the infrastructure and the logical levels. Probabilistic modeling was used to compare different design configurations, allowing the designers of the system to make choices that maximized the intrusion tolerance provided by the system before they actually implemented the system. Lastly, the model was used to show that the system would meet a set of quantitative survivability requirements.

9. CONCLUDING REMARKS

The last 50 years have witnessed the introduction of multiple new ideas in dependable and secure computing and their development at the University of Illinois. With a strong commitment to this area of research by a large staff of researchers, we expect to see many more exciting results in the future. New research in application-aware error detection and recovery, fault tolerance middleware, benchmarking of system dependability, dependability modeling, and secure system design and validation continue to maintain Illinois's status as a leading center in fault-tolerant and secure computing research.

ACKNOWLEDGEMENTS

We would like to thank our past and current research sponsors for supporting this work, and our colleagues, both past and present, that also contributed to the research described herein. Our sponsors include the Gigascale Systems Research Center (GSRC/MARCO), the NSF (NSF CCR 99-02026, NSF CCR 00-86096 ITR, NSF ACI 0121658 ITR/AP, NSF EIA 99-75019, NSF INT 0233490, NSF CCR 0311616), MURI Grant N00014-01-1-0576, the Defense Advanced Research Projects Agency, the Jet Propulsion Laboratory, the Semiconductor Research Corporation, Motorola, and Pioneer Hybrid. We would also like to thank Ms. Jenny Applequist for her editorial assistance.

REFERENCES

[1] J. Abraham, D. Gajski, "Design of Testable Structures Defined by Simple Loops," *IEEE Trans. on Comp.*, C-30, 1981, pp. 875-884

[2] J. Abraham, E. Davidson, J. Patel, "Memory System Design for Tolerating Single-Event Upsets," *IEEE Trans. on Nuclear Science*, NS-30, No. 6, 1983, pp. 4339-4344.

[3] J. Abraham, G. Metze, "Roving Diagnosis for High-Performance Digital Systems," *Proc. Conf. on Information Sciences and Systems*, 1978, pp. 221-226.

[4] N. Alewine, K. Fuchs, W-M. Hwu, "Application of Compiler-Assisted Multiple Instruction Rollback Recovery to Speculative Execution," Technical Report CRHC-93-16, University of Illinois, 1993.

[5] D. Anderson, "Design of Self-Checking Digital Networks," Coordinated Science Laboratory Technical Report R527, University of Illinois, Urbana, Illinois, 1971.

[6] D. Anderson, G. Metze, "Design of Totally Self-Checking Circuits for m-out-of-n Codes," *IEEE Trans. on Comp.*, C-22, No. 3, 1973, pp. 263-269.

[7] S. Bagchi, "Hierarchical Error Detection in a SIFT Environment," Ph.D. Thesis, University of Illinois, 2000.

[8] S. Bagchi, Y. Liu, Z. Kalbarczyk, R. K. Iyer, Y. Levendel, L. Votta, "A Framework for Database Audit and Control Flow Checking for a Wireless Telephone Network Controller," *Proc. of Conf. on Dependable Systems and Networks, DSN'01*, July 2001, pp. 225-234.

[9] P. Banerjee, J. Abraham, "Characterization and Testing of Physical Failures in MOS Logic Circuits," *IEEE Design and Test*, 1, 1984, pp. 76-86

[10] P. Banerjee, J. Abraham, "Fault-Secure Algorithms for Multiple Processor Systems," *Proc. 11th Int. Symp. on Computer Architecture*, 1984, pp. 279-287.

[11] P. Banerjee, J. Abraham, "A Multivalued Algebra for Modeling Physical Failures in MOS VLSI Circuits," *IEEE Trans. on Computer-Aided Design*, CAD-4, No. 3, 1985, pp. 312-321.

[12] P. Banerjee, J. Abraham, "Bounds on Algorithm-Based Fault Tolerance in Multiple Processor Systems," *IEEE Trans. on Comp.*, C-35, No. 4, 1986, pp. 296-306.

[13] H. Bohnenkamp, T. Courtney, D. Daly, S. Derisavi, H. Hermanns, J.-P. Katoen, R. Klaren, V. V. Lam, W. H. Sanders, "On Integrating the Möbius and Modest Modeling Tools," *Proc. Int. Conf. on Dependable Systems and Networks*, San Francisco, CA, June 22-25, 2003, p. 671.

[14] V. Boppana, W. K. Fuchs, "Fault Dictionary Compaction by Output Sequence Removal," *Proc. IEEE/ACM Conf. Computer-Aided Design*, Nov. 1994, pp. 576-579.

[15] W. Carter, P. Schneider, "Design of Dynamically Checked Computers," *Proc. IFIP Congress 2*, 1968, pp. 878-883.
[16] C. Cha, "Multiple Fault Diagnosis in Combinational Networks," Coordinated Science Laboratory Technical Report R-650, University of Illinois, Urbana, Illinois, 1974.
[17] R. Chandra, M. Cukier, R. M. Lefever, W. H. Sanders, "Dynamic Node Management and Measure Estimation in a State-Driven Fault Injector," *Proc. 19th IEEE Symp. on Reliable Distributed Systems*, Nürnberg, Germany, October 16-18, 2000, pp. 248-257.
[18] R. Chandra, R. M. Lefever, M. Cukier, W. H. Sanders, "Loki: A State-Driven Fault Injector for Distributed Systems," *Proc. Int. Conf. on Dependable Systems and Networks (DSN-2000)*, New York, NY, June 25-28, 2000, pp. 237-242.
[19] R. Chandra, R. M. Lefever, K. Joshi, M. Cukier, W. H. Sanders, "A Global-State-Triggered Fault Injector for Distributed System Evaluation," *IEEE Trans. on Par. and Dist. Systems*, to appear.
[20] H. Chang, E. Manning, C. Metze, *Fault Diagnosis of Digital Systems*, Huntington, NY: Robert E. Krieger Publishing Company, 1970.
[21] M. Chang, W. Fuchs, J. Patel, "Diagnosis and Repair of Memory with Coupling Faults," *IEEE Trans. Comp.*, April 1989, pp. 493-500.
[22] S. Chen, J. Xu, Z. Kalbarczyk, R. K. Iyer, K. Whisnant, "Modeling and Evaluating the Security Threats of Transient Errors in Firewall Software," *Int. Journal on Performance Evaluation*, 56, March 2004, pp. 53-72.
[23] S. Chen, K. Pattabiraman, Z. Kalbarczyk, R. K. Iyer, "Formal Reasoning of Various Categories of Widely Exploited Security Vulnerabilities Using Pointer Taintedness Semantics," *Proc. 19th IFIP Int'l Information Security Conf. (SEC-2004)*, held as part of 18th IFIP World Computer Congress, August 2004.
[24] S. Chen, Z. Kalbarczyk, J. Xu, R. K. Iyer, "A Data-Driven Finite State Machine Model for Analyzing Security Vulnerabilities," *Proc. Int. Conf. on Dependable Systems and Networks, DSN'03*, June 2003, pp. 605-614.
[25] W. Cheng, "The BACK Algorithm for Sequential Test Generation," *Proc. IEEE Int. Conf. on Computer Design*, Oct 1988, pp. 66-69.
[26] W. Cheng, J. Patel, "Concurrent Error Detection in Iterative Logic Arrays," *Proc. 14th Int. Symp. on Fault-Tolerant Computing*, 1984, pp. 10-15.
[27] W. Cheng, J. Patel, "A Minimum Test Set for Multiple-Fault Detection in Ripple-Carry Adders," *Proc. Int. Conf. on Computer Design*, 1985, pp. 435-438.
[28] W. Cheng, J. Patel, "Multiple-Fault Detection in Iterative Logic Arrays," *Proc. Int. Test Conf.*, 1985, pp. 493-499.
[29] W. Cheng, J. Patel, "A Shortest Length Test Sequence for Sequential-Fault Detection in Ripple Carry Adders," *Proc. Int. Conf. on Computer-Aided Design*, 1985, pp. 71-73.
[30] R. Chillarege, R. K. Iyer, "Fault Latency in the Memory: An Experimental Study on VAX 11/780," *Proc. 16th Int. Symp. on Fault-Tolerant Computing*, 1986.
[31] R. Chillarege, R. K. Iyer, "The Effect of System Workload on Error Latency: An Experimental Study," *Proc. ACM SIGMETRICS Conf. on Measurement and Modeling of Computer Systems*, 1985, pp. 69-77.
[32] A. L. Christensen, "Result Specification and Model Connection in the Möbius Modeling Framework," Master's Thesis, University of Illinois, 2000.
[33] G. Clark, T. Courtney, D. Daly, D. Deavours, S. Derisavi, J. M. Doyle, W. H. Sanders, P. Webster, "The Möbius Modeling Tool," *Proc. 9th Int. Workshop on Petri Nets and Performance Models*, Aachen, Germany, September 11-14, 2001, pp. 241-250.
[34] G. Clark, W. H. Sanders, "Implementing a Stochastic Process Algebra within the Möbius Modeling Framework," in Luca de Alfaro and Stephen Gilmore (Eds.), *Process Algebra and Probabilistic Methods: Performance Modelling and Verification: Proc. Joint Int. Workshop, PAPM-PROBMIV 2001*, RWTH Aachen, Germany, September 12-14, 2001, *LNCS* no. 2165, Berlin: Springer, 2001, pp. 200-215.

[35] T. Courtney, J. Lyons, H. V. Ramasamy, W. H. Sanders, M. Seri, M. Atighetchi, P. Rubel, C. Jones, F. Webber, P. Pal, R. Watro, M. Cukier, J. Gossett, "Providing Intrusion Tolerance with ITUA," *Supplemental Volume Int. Conf. on Dependable Systems & Networks (DSN-2002)*, Washington, DC, June 23-26, 2002, pp. C-5-1 to C-5-3.

[36] M. Cukier, R. Chandra, D. Henke, J. Pistole, W. H. Sanders, "Fault Injection Based on a Partial View of the Global State of a Distributed System," *Proc. 18th IEEE Symp. on Reliable Distributed Systems*, Lausanne, Switzerland, October 19-22, 1999, pp. 168-177.

[37] J. Cusey, J. Patel, "BART: A Bridging Fault Test Generator for Sequential Circuits," *Proc. Int. Test Conf.*, Nov. 1997, pp. 838-847.

[38] D. Daly, P. Buchholz, W. H. Sanders, "An Approach for Bounding Reward Measures in Markov Models Using Aggregation," submitted for publication.

[39] T. Davis, R. Kunda, K. Fuchs, "Testing of Bit-Serial Multipliers," *Proc. Int. Conf. on Computer Design*, 1985, pp. 430-434.

[40] D. D. Deavours, "Formal Specification of the Möbius Modeling Framework," Doctoral Dissertation, University of Illinois, 2001.

[41] D. D. Deavours, G. Clark, T. Courtney, D. Daly, S. Derisavi, J. M. Doyle, W. H. Sanders, P. G. Webster, "The Möbius Framework and Its Implementation," *IEEE Trans. on Software Eng.*, 28, No. 10, October 2002, pp. 956-969.

[42] D. D. Deavours, W. H. Sanders, "The Möbius Execution Policy," *Proc. 9th Int. Workshop on Petri Nets and Performance Models*, Aachen, Germany, September 11-14, 2001, pp. 135-144.

[43] D. D. Deavours, W. H. Sanders, "Möbius: Framework and Atomic Models," *ibid.*, pp. 251-260.

[44] S. Derisavi, P. Kemper, W. H. Sanders, "Symbolic State-space Exploration and Numerical Analysis of State-sharing Composed Models," *Proc. 4th Int. Conf. on the Numerical Solution of Markov Chains*, Urbana, IL, USA, Sept. 3-5, 2003, pp. 167-189.

[45] S. Derisavi, P. Kemper, W. H. Sanders, "Symbolic State-space Exploration and Numerical Analysis of State-sharing Composed Models," *Linear Algebra and Its Applications (LAA)*, to appear.

[46] S. Derisavi, P. Kemper, W. H. Sanders, T. Courtney, "The Möbius State-level Abstract Functional Interface," *Performance Evaluation*, 54, No. 2, October 2003, pp. 105-128.

[47] Y. Deswarte, L. Blain, J. C. Fabre, "Intrusion Tolerance in Distributed Computing Systems," *Proc. IEEE Symp. on Research in Security & Privacy*, May 1991, pp. 110-121.

[48] J. Dussault, "On the Design of Self-Checking Systems under Various Fault Models," Coordinated Science Lab. Technical Report R-781, Univ. of Illinois, Urbana, IL, 1977.

[49] B. Dutertre, V. Crettaz, V. Stavridou, "Intrusion-tolerant Enclaves," *Proc. IEEE Int. Symp. on Security & Privacy*, Oakland, CA, May 2002, pp. 216–224.

[50] A. Friedman, L. Simoncini, "System-Level Fault Diagnosis," *Computer* (Spec. Issue on Fault-Tolerant Computing), 13, No. 3, 1980, pp. 47-53.

[51] K. Goswami, R. K. Iyer, L. Young, "DEPEND: A Simulation-Based Environment for System Level Dependability Analysis," *IEEE Trans. on Comp.*, vol. 46, no. 1, 1997, pp.60-74.

[52] G. Greenstein, J. Patel, "E-PROOFS: A CMOS Bridging Fault Simulator," *Proc. IEEE/ACM Conf. on Computer-Aided Design*, Nov. 1992, pp. 268 – 271.

[53] W. Gu, Z. Kalbarczyk, R. K. Iyer, "Error Sensitivity of the Linux Kernel Executing on PowerPC G4 and Pentium 4 Processors," *Proc. Conf. on Dependable Systems & Networks, DSN'04*, June 2004.

[54] W. Gu, Z. Kalbarczyk, R.K. Iyer, Z. Yang, "Characterization of Linux Kernel Behavior under Errors," *Proc. Conf. on Dependable Systems & Networks, DSN'03*, June 2003, pp. 459-468.

[55] J. Gunnels, D. Katz, E. Quintana-Orti, R. van de Geijn, "Fault-Tolerant High-Performance Matrix Multiplication: Theory and Practice," *Proc. Conf. on Dependable Systems and Networks, DSN'01*, July 2001, pp. 47-56.

[56] V. Gupta, V. Lam, H. V. Ramasamy, W. H. Sanders, S. Singh, "Dependability and Performance Evaluation of Intrusion-Tolerant Server Architectures," *Dependable Computing: Proc. First Latin-American Symposium (LADC 2003)*, São Paulo, Brazil, October 21-24, 2003, *LNCS* vol. 2847 (Rogério de Lemos, Taisy Silva Weber, and João Batista Camargo Jr., eds), Berlin: Springer, 2003, pp. 81-101.
[57] I. Hamzaoglu, J. Patel, "Test Set Compaction Algorithms for Combinational Circuits," *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, Nov. 1998, pp. 283-289.
[58] I. Hamzaoglu, J. Patel, "Reducing Test Application Time for Full Scan Embedded Cores," Proc. *29th Int. Symp. on Fault-Tolerant Computing*, June 1999, pp. 260-267.
[59] I. Hartanto, V. Boppana, W. Fuchs, J. Patel, "Diagnostic Test Generation for Sequential Circuits," *Proc. IEEE VLSI Test Symp*., April 1997, pp. 196-202.
[60] J. Hayes, "A NAND Model for Fault Diagnosis in Combinational Logic Networks," *IEEE Trans. on Comp.*, C-20, 1971, pp. 1496-1506.
[61] F. Hsu, K. Butler, J. Patel, "A Case Study on the Implementation of the Illinois Scan Architecture," *Proc. Int. Test Conf.*, Oct-Nov 2001, pp. 538-547.
[62] http://www.securityfocus.com.
[63] K. Huang, J. Abraham, "Low-Cost Schemes for Fault Tolerance in Matrix Operations with Array Processors," *Proc. 12th Int. Symp. on Fault-Tolerant Computing*, 1982, pp. 330-337.
[64] K. Huang, J. Abraham, "Algorithm-Based Fault Tolerance for Matrix Operations," *IEEE Trans. on Comp.* (Spec. Issue Reliable & Fault-Tolerant Comp.), C-33, 1984, pp. 518-528.
[65] K. Huang, J. Abraham, "Fault-Tolerant Algorithms and their Applications to Solving Laplace Equations," *Proc. Int. Conf. on Parallel Processing*, 1984, pp. 117-122.
[66] R. K. Iyer, D. Rossetti, "A Measurement-Based Model for Workload Dependency of CPU Errors," *IEEE Trans. on Comp.*, C-35, No. 6, 1986.
[67] R. K. Iyer, D. Rossetti, "Effect of System Workload on Operating System Reliability: A Study on the IBM 3081," *IEEE Trans. on Software Eng.* (Spec. Issue Software Reliability, Part 1), 1985, pp. 1438-1448.
[68] I. Jansch, B. Courtois, "Strongly Language Disjoint Checkers," *Proc. 15th Int. Symp. on Fault-Tolerant Computing*, 1985, pp. 390-395.
[69] H. S. Javitz, A. Valdes, "The SRI IDES Statistical Anomaly Detector," *Proc. IEEE Symp. on Research in Security and Privacy*, Oakland, CA, May 1991, pp. 316-376.
[70] K. R. Joshi, M. Cukier, W. H. Sanders, "Experimental Evaluation of the Unavailability Induced by a Group Membership Protocol," *Dependable Computing EDCC-4: Proc. 4th European Dependable Computing Conf.*, Toulouse, France, Oct 23-25, 2002, pp. 140-158.
[71] J. Jou, J. Abraham, "Fault-Tolerant Matrix Operations on Multiple Processor Systems Using Weighted Checksums," *Proc. SPIE Conf.*, 1984, pp. 94-101.
[72] J. Jou, J. Abraham, "Fault-Tolerant FFT Networks," *Proc. Int. Symp. on Fault-Tolerant Computing*, 1985, pp. 338-343.
[73] Z. Kalbarczyk, R. K. Iyer, S. Bagchi, K. Whisnant, "Chameleon: A Software Infrastructure for Adaptive Fault Tolerance," *IEEE Trans. on Par. and Dist. Systems*, 10, No. 6, June 1999, pp. 560-579.
[74] M. Kalyanakrishnam, R. K. Iyer, J. Patel, "Reliability of Internet Hosts: A Case Study from End User's Perspective," *Proc. 6th Int. Conf. on Computer Communications and Networks*, Las Vegas, 1996, pp. 418-423.
[75] M. Kalyanakrishnam, Z. Kalbarczyk, R. Iyer, "Failure Data Analysis of LAN of Windows NT Based Computers," *Proc. of 18th Symp. on Reliable and Distributed Systems, SRDS '99*, Lausanne, Switzerland, 1999, pp. 178-187.
[76] S. Krishnamurthy, "An Adaptive Quality of Service Aware Middleware for Replicated Services," Ph.D. Thesis, University of Illinois, 2002.
[77] S. Krishnamurthy, W. H. Sanders, M. Cukier, "An Adaptive Quality of Service Aware Middleware for Replicated Services," *IEEE Trans. on Par. and Dist. Systems*, 14, No. 11, November 2003, pp. 1112-1125.

[78] S. Kuo, K. Fuchs, “Efficient Spare Allocation in Reconfigurable Arrays,” *IEEE Design and Test*, Feb. 1987, pp. 24-31.

[79] S. Laha, J. Patel, “Error Correction in Arithmetic Operations using Time Redundancy,” *Proc. 13th Int. Symp. on Fault-Tolerant Computing*, 1983, pp. 298-305.

[80] V. V. Lam, P. Buchholz, W. H. Sanders, “A Structured Path-Based Approach for Computing Transient Rewards of Large CTMCs,” *Proc. 2004 Int. Conf. on Quantitative Evaluation of Systems (QEST)*, Twente, The Netherlands, September 27-30, 2004.

[81] C. Landwehr, “Formal Models for Computer Security,” *Computer Surveys*, 13, No. 3, Sept. 1981.

[82] R. M. Lefever, M. Cukier, W. H. Sanders, “An Experimental Evaluation of Correlated Network Partitions in the Coda Distributed File System,” *Proc. 22nd Int. Symp. on Reliable Distributed Systems (SRDS'03),* Florence, Italy, October 6-8, 2003, pp. 273-282.

[83] B. Littlewood, S. Brocklehurst, N. Fenton, P. Mellor, S. Page, D. Wright, J. Dobson, J. McDermid, D. Gollmann, “Towards Operational Measures of Computer Security,” *Journal of Computer Security*, vol. 2, no. 2-3, pp. 211-229, 1993.

[84] J. Lowry, “An Initial Foray into Understanding Adversary Planning and Courses of Action,” *Proc. DARPA Information Survivability Conf. and Exposition II (DISCEX'01)*, pp. 123-133, 2001.

[85] F. Luk, “Algorithm-Based Fault Tolerance for Parallel Matrix Equation Solvers,” *Proc. SPIE Conf. (Real-Time Signal Processing VIII),* 564, 1985.

[86] L. Malhis, “Development and Application of an Efficient Method for the Solution of Stochastic Activity Networks with Deterministic Activities,” Doctoral Dissertation, University of Arizona, 1996.

[87] L. Malhis and W. H. Sanders, “An Efficient Two-Stage Iterative Method for the Steady-State Analysis of Markov Regenerative Stochastic Petri Net Models,” *Performance Evaluation*, 27&28, October 1996, pp. 583-601.

[88] E. Manning, “On Computer Self-Diagnosis: Part I and II,” *IEEE Trans. Electronic Comp.*, EC-15, 1966, pp. 873-890.

[89] R. Marlett, “An Effective Test Generation System for Sequential Circuits,” *Proc. Design Automation Conf.*, June 1986, pp. 250-256.

[90] R. Marlett, “On the Design and Testing of Self-Diagnosable Computers,” Coordinated Science Laboratory Technical Report R-293, University of Illinois, Urbana, IL, 1966.

[91] E. McCluskey, F. Clegg, “Fault Equivalence in Combinational Logic Networks,” *IEEE Trans. on Comp.*, C- 20, 1971, pp. 1286-1293.

[92] R. Meagher, J. Nash, “The ORDVAC,” *Review of Electronic Digital Computers*, 1952, pp. 37-43.

[93] D. Muller, J. Bartky, “A Theory of Asynchronous Circuits,” *Proc. Int. Symp. on Theory of Switching*, 1959, pp. 204-243.

[94] R. Nair, S. Thatte, J. Abraham, “Efficient Algorithms for Testing Semiconductor Random-Access Memories,” *IEEE Trans. on Comp.*, C-27, No. 6, 1978, pp. 572-576.

[95] N. Nakka, J. Xu, Z. Kalbarczyk, R. K. Iyer, “An Architectural Framework for Providing Reliability and Security Support,” *Proc. Conf. on Dependable Systems and Networks, DSN’04*, June 2004.

[96] N. Neves, K. Fuchs, “Coordinated Checkpointing Without Direct Coordination,” *Proc. Int. Computer Performance and Dependability Symp. (IPDS),* 1998.

[97] N. Neves, K. Fuchs, “RENEW: A Tool for Fast and Efficient Implementation of Checkpoint Protocols,” *Proc. 28th Int. Symp. on Fault-Tolerant Computing*, 1998, pp.58-67.

[98] T. Niermann, J. Patel, “HITEC: A Test Generation Package for Sequential Circuits,” *Proc. European Design Automation Conf.*, Feb. 2001, pp. 214-218.

[99] T. Niermann, W. Cheng and J. Patel, “PROOFS: A Fast Memory Efficient Sequential Circuit Fault Simulator,” *IEEE Trans. On Computer-Aided Design*, Feb. 1992, pp. 198-207.

[100] W. D. Obal II, “Measure-Adaptive State-Space Construction Methods,” Doctoral Dissertation, University of Arizona, 1998.

[101] W. D. Obal II, W. H. Sanders, “An Environment for Importance Sampling Based on Stochastic Activity Networks,” *Proc. 13th Symp. on Reliable Distributed Systems*, Dana Point, CA, October, 1994, pp. 64-73.

[102] W. D. Obal II, W. H. Sanders, “State-Space Support for Path-based Reward Variables,” *Proc. Int. Computer Performance and Dependability Symp. (IPDS)*, September 7-9, 1998, Durham, North Carolina, USA, pp. 228-237.

[103] W. D. Obal II, W. H. Sanders, “State-Space Support for Path-based Reward Variables,” *Performance Evaluation*, 35, 1999, pp. 233-251.

[104] P. Pal, F. Webber, R. Schantz, J. Loyall, R. Watro, W. Sanders, M. Cukier, J. Gossett, “Survival by Defense-Enabling,” *Proc. New Security Paradigms Workshop 2001*, Cloudcroft, New Mexico, September 11-13, 2001, pp. 71-78.

[105] J. Patel, L. Fung, “Concurrent Error Detection in ALUs by Recomputing with Shifted Operands,” *IEEE Trans. on Comp.*, C-31, 1982, pp. 589-595.

[106] J. Patel, L. Fung, “Concurrent Error Detection in Multiply and Divide Arrays,” *IEEE Trans. on Comp.*, C-32, 1983, pp. 417-422.

[107] C. Payne, T. Markham, “Architecture and Applications for a Distributed Embedded Firewall,” *Proc. 17th Annual Computer Security Applications Conf. (ACSAC’01)*, New Orleans, Louisiana, October 2001.

[108] L. Pollard and J. Patel, “Correction of Errors in Data Transmission using Time Redundancy,” *Proc. 13th Int. Symp. on Fault-Tolerant Computing*, 1983, pp. 314-317.

[109] F. Preparata, G. Metze, R. Chien, “On the Connection Assignment Problem of Diagnosable Systems,” *IEEE Trans. on Electronic Comp.*, EC-16, No. 6, 1967, pp. 848-854.

[110] M. A. Qureshi, “Construction and Solution of Markov Reward Models,” Doctoral Dissertation, University of Arizona, 1996.

[111] M. A. Qureshi, W. H. Sanders, “A New Methodology for Calculating Distributions of Reward Accumulated During a Finite Interval,” *Proc. 26th Int. Symp. on Fault-Tolerant Computing*, Sendai, Japan, June 1996, pp. 116-125.

[112] H. V. Ramasamy, M. Cukier, W. H. Sanders, “Formal Verification of an Intrusion-Tolerant Group Membership Protocol,” *IEICE Trans. on Information and Systems* Spec. issue on Dependable Computing, E86-D, No. 12, December 2003, pp. 2612-2622.

[113] H.V. Ramasamy, P. Pandey, J. Lyons, M. Cukier, W.H. Sanders, “Quantifying the Cost of Providing Intrusion Tolerance in Group Communication Systems,” *Proc. Int. Conf. on Dependable Systems & Networks (DSN’02)*, Washington, DC, June 23-26, 2002, pp. 229-238.

[114] J. Rearick, J. Patel, “Fast and Accurate CMOS Bridging Fault Simulation,” *Proc. Int. Test Conf.*, Oct. 1993, pp. 54-62.

[115] Y. Ren, “AQuA: A Framework for Providing Adaptive Fault Tolerance to Distributed Applications,” Ph.D. thesis, University of Illinois at Urbana-Champaign, 2001.

[116] Y.(J.) Ren, D.E. Bakken, T. Courtney, M. Cukier, D.A. Karr, P. Rubel, C. Sabnis, W.H. Sanders, R.E. Schantz, M. Seri, “AQuA: An Adaptive Architecture that Provides Dependable Distributed Objects,” *IEEE Trans. on Comp.*, 52, No. 1, January 2003, pp. 31-50.

[117] Y. Ren, M. Cukier, W.H. Sanders, “An Adaptive Algorithm for Tolerating Value Faults and Crash Failures,” *IEEE Trans. on Par. & Dist. Systems*, 12, No. 2, Feb. 2001, pp. 173-192.

[118] D. Reynolds, C. Metze, “Fault Detection Capabilities of Alternating Logic,” *IEEE Trans. on Comp.*, C-27, 1978, pp. 1093-1098.

[119] P. Ryan, S. Rawat, W. Fuchs, “Two-stage Fault Location,” *Proc. Int. Test Conf.*, Oct. 1991, pp. 963-968.

[120] C. Sabnis, M. Cukier, J. Ren, P. Rubel, W. H. Sanders, D. E. Bakken, D. A. Karr, “Proteus: A Flexible Infrastructure to Implement Adaptive Fault Tolerance in AQuA,” in C. B. Weinstock and J. Rushby (Eds.), *Dependable Computing for Critical Applications 7*, vol. 12 in *Dependable Computing and Fault-Tolerant Systems* (A. Avizienis, H. Kopetz, and J. C. Laprie, Eds.), pp. 149-168. Los Alamitos, CA: IEEE CS, 1999.

[121] A. Saleh, J. Serrano, J. Patel, “Reliability of Scrubbing Recovery Techniques for Memory Systems,” *IEEE Trans. on Reliability*, April 1990, pp. 114-122.

[122] W. H. Sanders, “Construction and Solution of Performability Models Based on Stochastic Activity Networks,” Doctoral Dissertation, University of Michigan, 1988.

[123] W. H. Sanders, “Integrated Frameworks for Multi-Level and Multi-Formalism Modeling,” *Proc. of PNPM’99: 8th Int. Workshop on Petri Nets and Performance Models*, Zaragoza, Spain, September 8-10, 1999, pp. 2-9.

[124] W. H. Sanders, M. Cukier, F. Webber, P. Pal, R. Watro, “Probabilistic Validation of Intrusion Tolerance,” *Supplemental Volume Int. Conf. on Dependable Systems & Networks (DSN-2002)*, Washington, DC, June 23-26, 2002, pp. B-78 to B-79.

[125] W. H. Sanders, R. S. Freire, “Efficient Simulation of Hierarchical Stochastic Activity Network Models,” *Discrete Event Dynamic Systems: Theory and Applications*, 3, No. 2/3, July 1993, pp. 271-300.

[126] W. H. Sanders, J. F. Meyer, “Reduced Base Model Construction Methods for Stochastic Activity Networks,” *IEEE Journal on Selected Areas in Communications*, special issue on *Computer-Aided Modeling, Analysis, and Design of Communication Networks*, vol. 9, no. 1, Jan. 1991, pp. 25-36.

[127] W. H. Sanders, J. F. Meyer, “Stochastic Activity Networks: Formal Definitions and Concepts,” in E. Brinksma, H. Hermanns, and J. P. Katoen (Eds.), *Lectures on Formal Methods and Performance Analysis, First EEF/Euro Summer School on Trends in Computer Science, Berg en Dal, The Netherlands, July 3-7, 2000, Revised Lectures, LNCS* no. 2090, pp. 315-343. Berlin: Springer, 2001.

[128] W. H. Sanders, J. F. Meyer, “A Unified Approach for Specifying Measures of Performance, Dependability, and Performability,” *Dependable Computing for Critical Applications,* Vol. 4 of *Dependable Computing and Fault-Tolerant Systems* (ed. A. Avizienis, H. Kopetz, and J. Laprie), Springer-Verlag, 1991, pp. 215-237.

[129] W.H. Sanders, W.D. Obal II, M.A. Qureshi, F.K. Widjanarko, “The *UltraSAN* Modeling Environment,” *Performance Evaluation*, 24, No. 1, Oct.-Nov. 1995, pp. 89-115.

[130] D. Schertz, G. Metze, “A New Representation for Faults in Combinational Digital Circuits,” *IEEE Trans. on Comp.*, C-21, No. 8, 1972, pp. 858-866.

[131] D. Schertz, “On the Representation of Digital Faults,” Coordinated Science Laboratory Technical Report R418, University of Illinois, Urbana, Illinois, 1969.

[132] D. Schertz, G. Metze, “On the Indistinguishability of Faults in Digital Systems,” *Proc. 6th Ann. Allerton Conf. on Circuit and System Theory*, 1968, pp. 752-760.

[133] M. Seri, T. Courtney, M. Cukier, V. Gupta, S. Krishnamurthy, J. Lyons, H. Ramasamy, J. Ren, W. H. Sanders, “A Configurable CORBA Gateway for Providing Adaptable System Properties,” *Supplemental Volume Int. Conf. on Dependable Systems & Networks (DSN-2002)*, Washington, DC, June 23-26, 2002, pp. G-26 to G-30.

[134] S. Seshu, D. Freeman, “The Diagnosis of Asynchronous Sequential Switching Systems”, *IRE Trans. on Electronic Comp.*, EC-11, No.4, 1962, pp. 459-465.

[135] S. Seshu, “The Logic Organizer and Diagnosis Programs”, Coordinated Science Laboratory Technical Report R226, University of Illinois, Urbana, IL, 1964.

[136] S. Seshu, “On an Improved Diagnosis Program”, *IEEE Trans. on Electronic Comp. EC-14*, No. 1, 1965, pp. 76-79.

[137] S. Singh, M. Cukier, W. H. Sanders, “Probabilistic Validation of an Intrusion-Tolerant Replication System,” *Proc. Int. Conf. on Dependable Systems and Networks (DSN-2003)*, San Francisco, CA, June 22-25, 2003, pp. 615-624.

[138] J. Smith, “On Necessary and Sufficient Conditions for Multiple Fault Undetectability,” *IEEE Trans. on Comp.*, C-28, 1979, pp. 801-802

[139] J. Smith, “The Design of Totally Self-Checking Combinational Circuits,” Coordinated Science Laboratory Technical Report R-737, University of Illinois, Urbana, IL, 1976.

[140] J. Smith, G. Metze, “On the Existence of Combinational Networks with Arbitrary Multiple Redundancies,” Coordinated Science Laboratory Technical Report R-692, University of Illinois, Urbana, IL, 1975.

[141] J. Smith, G. Metze, "Strongly Fault-Secure Logic Networks," *IEEE Trans. on Comp.*, C-27, No. 6, 1978, pp. 491-499.

[142] F. Stevens, "Validation of an Intrusion-Tolerant Information System Using Probabilistic Modeling," Master's Thesis, University of Illinois, 2004.

[143] D. Stott, B. Floering, D. Burke, Z. Kalbarczyk, R. K. Iyer, "NFTAPE: A Framework for Assessing Dependability in Distributed Systems with Lightweight Fault Injectors," *Proc. Int. Computer Performance & Dependability Symp. (IPDS)*, March 2000, pp. 91-100.

[144] D. Stott, P. Jones, M. Hamman, Z. Kalbarczyk, R. Iyer, "NFTAPE: Network Fault Tolerance and Performance Evaluator," *Proc. Int. Conf. on Dependable Systems and Networks, DSN'02*, June 2002, pp. 542.

[145] D. Suk, S. Reddy, "A March Test for Functional Faults in Semiconductor Random Access Memories," *IEEE Trans. on Comp.*, C-30, 1981, pp. 982-984.

[146] S. Thatte, J. Abraham, "Testing of Semiconductor Random Access Memories," *Proc. 7th Int. Symp. on Fault-Tolerant Computing*, 1977, pp. 81-87.

[147] S. Thatte, J. Abraham, "Test Generation for Microprocessors," *IEEE Trans. on Comp.*, C-29, No. 6, 1980, pp. 429-441.

[148] K. To, "Fault Folding for Irredundant and Redundant Combinational Circuits", *IEEE Trans. on Comp. C-22*, No. 11, 1973, pp. 1008-1015.

[149] T. Tsai, M-Ch. Hsueh, H. Zhao, Z. Kalbarczyk, R. K. Iyer, "Stress-based and Path-based Fault Injection," *IEEE Trans. on Comp.*, 48, No. 11, Nov. 1999, pp. 1183-1201.

[150] J. Tvedt, "Solution of Large-Sparse Stochastic Process Representations of Stochastic Activity Networks," Master's Thesis, University of Arizona, 1990.

[151] E. Ulrich and T. Baker, "The Concurrent Simulation of Nearly Identical Digital Systems," *Proc. 10th Design Automation Workshop*, June 1973, pp. 145-150.

[152] A. P. A. van Moorsel, W. H. Sanders, "Adaptive Uniformization," *ORSA Communications in Statistics: Stochastic Models*, 10, No. 3, August 1994, pp. 619-648.

[153] A. P. A. van Moorsel, W. H. Sanders, "Transient Solution of Markov Models by Combining Adaptive & Standard Uniformization," *IEEE Trans. on Reliability*, 46, No. 3, September 1997, pp. 430-440.

[154] L. Wang, Z. Kalbarczyk, R. K. Iyer, H. Vora, T. Chahande, "Checkpointing of Control Structures in Main Memory Database Systems," *Proc. Conf. on Dependable Systems and Networks, DSN'04*, June 2004.

[155] Y. Wang, K. Fuchs, "Optimal Message Log Reclamation for Independent Checkpointing," Technical Report CRHC-93-07, University of Illinois, 1993.

[156] D. Wheeler and J. Robertson, "Diagnostic Programs for the ILLIAC," *Proc. IRE 41*, 1953, pp. 1320-1325.

[157] K. Whisnant, R.K. Iyer, Z. Kalbarczyk, P.H. Jones III, D.A. Rennels, R. Some, "The Effects of an ARMOR-Based SIFT Environment on the Performance and Dependability of User Applications," *IEEE Trans. on Software Eng.*, 30, No. 4, April 2004, pp. 257-277.

[158] K. Whisnant, Z. Kalbarczyk, R. Iyer, "A System Model for Reconfigurable Software," *IBM Systems Journal*, 42, No. 1, 2003, pp. 45-59.

[159] J. Xu, S. Chen, Z. Kalbarczyk, R. K. Iyer, "An Experimental Study of Security Vulnerabilities Caused by Errors," *Proc. Conf. on Dependable Systems and Networks, DSN'01*, July 2001, pp. 421-430.

[160] J. Xu, Z. Kalbarczyk, R.K. Iyer, "Transparent Runtime Randomization for Security," *Proc. Symp. on Reliable and Distributed Systems, SRDS'03*, October 2003.

[161] J. Xu, Z. Kalbarczyk, R. Iyer, "Networked Windows NT System Filed Failure Data Analysis," *Proc. Pacific Rim Int. Symp. on Dependable Computing*, Hong Kong, 1999.