NASA/TM–20210013688

Establishing Fault Tolerance for a Class of Systems by Experiment

Allan L White
Langley Research Center, Hampton, Virginia

June 2021

NASA STI Program Report Series

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA scientific and technical information (STI) program plays a key part in helping NASA maintain this important role.

The NASA STI program operates under the auspices of the Agency Chief Information Officer. It collects, organizes, provides for archiving, and disseminates NASA's STI. The NASA STI program provides access to the NTRS Registered and its public interface, the NASA Technical Reports Server, thus providing one of the largest collections of aeronautical and space science STI in the world. Results are published in both non-NASA channels and by NASA in the NASA STI Report Series, which includes the following report types:

- TECHNICAL PUBLICATION. Reports of completed research or a major significant phase of research that present the results of NASA Programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA counterpart of peer-reviewed formal professional papers but has less stringent limitations on manuscript length and extent of graphic presentations.

- TECHNICAL MEMORANDUM. Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.

- CONTRACTOR REPORT. Scientific and technical findings by NASA-sponsored contractors and grantees.

- CONFERENCE PUBLICATION. Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or co-sponsored by NASA.

- SPECIAL PUBLICATION. Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.

- TECHNICAL TRANSLATION. English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services also include organizing and publishing research results, distributing specialized research announcements and feeds, providing information desk and personal search support, and enabling data exchange services.

For more information about the NASA STI program, see the following:

- Access the NASA STI program home page at http://www.sti.nasa.gov

- Help desk contact information:

https://www.sti.nasa.gov/sti-contact-form/
and select the "General" help request type.

NASA/TM–20210013688

Establishing Fault Tolerance for a Class of Systems by Experiment

Allan L White
Langley Research Center, Hampton, Virginia

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia
23681-2199

June 2021

Available from:

NASA STI Program / Mail Stop 148
NASA Langley Research Center
Hampton, VA 23681-2199
Fax: 757-864-6500

Abstract

A long-standing problem in system verification is establishing fault tolerance at the ultra-high level by experiment. It is considered impossible because of system complexity and the enormous number of trials needed. This paper considers the problem for a class of digital systems that use redundancy to achieve reliability. The class is the systems that operate for a period of time without maintenance followed by a maintenance check that replaces components identified as faulty. The paper considers simulating a natural life test where a natural life test observes a number of operating periods. If the system does not fail during the test, it can be said to have a certain reliability at a certain confidence level. The approach in this paper is to make the simulated life test more efficient while maintaining realism by integrating structural arguments, information on fault occurrence, and fault injection in the lab. The major result of this paper is constructing a global fault model using the failure rate of the components and proving theorems about the model that tell how many, what kind, when, and where to inject faults. A simple example illustrates applying the theorems.

I. INTRODUCTION

A. Objective

We propose a method of verifying fault tolerance at the ultra-high level by experiment for a class of systems and faults. We consider microprocessors for control and monitoring that use redundancy for reliability. They may or may not use reconfiguration. These systems have an operating period with no maintenance followed by a maintenance check where all components detected as faulty are replaced. The approach is simulated life testing where each trial represents an operating period. During a trial, we inject faults into the system based on the distributions for fault occurrence. We also use structural arguments that a system operates correctly if all the components are functioning correctly. Hence, the approach integrates data on fault occurrence, structural arguments, and laboratory experiments.

There are a number of aspects to reliability. One division is validation and verification where validation asks if we built the right thing and verification asks if we built it right. There are structural arguments that consider such topics as whether or not the system will work correctly if all the components are fault free. There are stochastic investigations that examine the effect of noise and fault occurrence on the system. This paper considers the probability of the system surviving fault occurrence.

B. Background and Motivation

The use of computers as controllers and monitors is becoming more widespread [1, 2], and some applications are safety critical [2, 3, 4]. A National Research Council document states that the lack of verification methods is a barrier to upgrading the National Airspace [2]. There is a desire for more performance-based, empirical demonstrations of reliability [38]. One topic is a

demonstration of fault tolerance at the ultra-high level. The obstacles are system complexity and the enormous number of trials necessary to establish fault tolerance at a high confidence level. This has been an open problem for fifty years and considered impossible for thirty years [4, 5, 6]. The sections below offer a solution.

C. Outline

Section two examines some of the literature in the field. It begins with references stating the growing use of microcontrollers and the importance of their reliability. It continues with references describing the difficulty of establishing ultra-reliability by experiment. Since there is a large amount of material on fault injection, the bibliography next considers the activity in the last decade seeking any evidence of the existence of any method of verification. This is followed by numerous papers describing efforts to characterize faults and inject them efficiently. The premise is that fault injection yields information that can improve system reliability.

Section three considers current approaches to verification while section four compares this paper's definition of a fault with another popular definition. Section five outlines the experimental approach. Section six presents the fault model and derives the theorems needed to apply it. Section seven describes the structure of a reconfigurable fourplex while the eighth applies the theorems in section six to design a fault injection experiment. Section nine illustrates integrating the experiment with an argument-from-design. Section ten considers extensions of the results in this paper.

II. BIBLIOGRAPHY

This bibliography describes the field of fault injection by topics: the importance of fault tolerance but the impossibility of establishing ultra-high fault tolerance by experiment, the problem of fault detection, using fault detection to test design, and the efficient injection of faults. This paper does not use any technical content of any of these papers. If the project continues, it may incorporate some of the results below.

A number of papers mention the importance of fault tolerance but the impossibility of establishing ultra-high fault tolerance by experiment. These papers provide the motivation for this endeavor. A publication [1] observes that digital controllers will become more prevalent. Another [2] lists the benefits of autonomous flight, it states that amongst the four highest priorities are verification and validation. A report [3] states that appropriate methods for assessing safety and reliability are key to establishing the acceptability of digital information and control systems in safety-critical plants. A chapter in a compendium [4] states that decades of theoretical and experimental work and numerous recent successful applications have established fault tolerance as a standard objective in computer system design, but in contrast to the objective of high speed, satisfaction of fault-tolerance requirements cannot be demonstrated by testing alone, but requires formal analysis. A publication [5] observes that safety-critical systems play an important role in modern societies, with increasing numbers of applications in many domains like transportation, nuclear energy and healthcare, but how to assess systems that require very high reliability remains a challenging task after almost 30 years since the first publications to

highlight the problem. A paper [6] concludes the insertion of artificial faults into a real or simulated computer system yields benefits, but to evaluate a system's fault tolerance capabilities is a much harder task, impossible even. An article [7] makes the case that greater reliance on computers in a variety of applications implies the consequences of failure have become more severe, but fault injection to determine a system's response offers an effective solution to this problem. The article surveys several fault-injection studies and tools. In a paper [8], the authors argue that early validation of fault tolerance is essential in developing dependable computer systems, and they defined a strategy for testing fault tolerance.

An important factor in reliability is the ability of a system to detect faults, called the diagnostic level, and recover from them. Otherwise, faults can accumulate and overwhelm the fault handling capabilities. An extensive effort [9] converts methods of simulating faults into injecting faults into hardware to study system properties and to estimate the diagnostic level. A consideration of memory errors [10] seeks to reduce the area and power requirement of error correcting codes without affecting delay in detection. In order to tolerate faults that emerge in operating Networks-on-Chip, diagnosis techniques are employed for fault detection and localization, and a paper [11] offers an improvement in diagnostics by combining different techniques. Field-Programmable Gate Arrays are rapidly gaining popularity as implementation platforms, and a paper [12] presents a method for transient and permanent fault mitigation and run-time fault recovery. Systems that are used in high dependability and integrity applications need to be designed with the capability to on-line detect and recover from the errors caused by faults, and since the majority of errors are usually transient and not reproducible, a paper [13] presents a set of system-level checks comprising for the on-line detection of errors. A paper [14] presents an approach to estimate the fault coverage of the implementation of a VLSI design obtained by fault simulation at the function level, and the results show a good correlation between the estimated fault coverage, based on fault simulation at the functional level, and the actual fault coverage obtained by fault simulation on a gate level implementation. A paper [15] discusses an experimental methodology for simulation-based validation of fault tolerant microprocessor architectures. It injects transient faults, and estimates the diagnostic level.

Fault injection has been used to test design features. An effort was made to uncover defects in system design by fault injection [16], but the effort revealed that random fault injection was not an efficient approach. A paper [17] observes that transient and permanent faults have been deeply studied while this work goes a step further and assess the dependability of a fault-tolerant computer system against intermittent faults. The effect of single event transient on reliability has become a significant concern for digital circuits, and a paper [18] proposes an algorithm for evaluating the reliability of digital circuits given their occurrence. Redundant execution is a common approach to achieve fault-tolerance, but a paper [19] observes it is energy inefficient and proposes a more efficient approach. The effect of compensating module faults on the reliability of majority voting based VLSI fault-tolerant circuits is investigated [20] using a fault injection simulation method where the simulation method facilitates consideration of multiple faults in the replicated circuit modules as well as the majority voting circuits. A paper [21] gives a brief description of a teraflops supercomputer and assesses signal sensitivity to transient faults and the effectiveness of the fault/error handling mechanisms by fault injection.

There is considerable effort with the goal of efficiently injecting faults. Optimized and custom arithmetic circuits are widely used in embedded systems, and a paper [22] presents an automated test generation and bug localization technique for debugging arithmetic circuits with experimental results demonstrating that the proposed approach can be used for automated debugging of large and complex arithmetic circuits. To achieve high reliability in on-chip networks, it is necessary to test the network continuously with Built-in Self-Tests so that the faults can be detected quickly and the number of affected packets can be minimized, but since built-in-self-tests cause significant performance loss due to data dependencies, a paper [23] proposes a comprehensive test strategy with minimized influence on system performance. Since efficient handling of faults during operation is highly dependent on the interval from the time embedded monitoring instruments detect faults to the time when the fault manager localizes the faults, an article [24] proposes a self-reconfiguring network in which all instruments that have detected faults are automatically included in the scan path, and a fault detection and localization module in hardware that detects the configuration of the network after self-reconfiguration and extracts the error codes reported by the fault monitoring instruments. A paper [25] presents a new fault injection method to increase the fault emulation performance by converting the original circuit into a fault-injectable circuit mapping the fault-injectable circuit onto the emulator. A paper [26] proposes three new techniques that substantially reduce the parallel fault simulation time where the new techniques are 1) reduction of faults simulated in parallel through mapping non-stem faults to stem faults, 2) a new fault injection method called functional fault injection, and 3) a combination of a static fault ordering method and a dynamic fault ordering method. A paper [27] addresses the problem of simulating and generating tests for transition faults in non scan or partial scan sequential circuits by enhancing the transition fault model for the gate-delay faults and the stuck-open faults in synchronous sequential circuits. Since the size and complexity of modern dependable computing systems has significantly compromised the ability to accurately measure system dependability attributes such as fault coverage and fault latency and fault injection techniques are difficult to apply because the size of the fault set is essentially infinite, a research effort [28] has developed a new deterministic, automated dependability evaluation technique using fault injection to yield more useful information. To maximize the efficacy of fault injection, a paper [29] presents two fault injection methodologies: stress-based injection and path-based injection. A paper [30] notes that ultra-dependable computing demands verification of fault-tolerant mechanisms in the hardware, and this research tries to bridge that gap by developing a new fault-injection methodology for processors based on a register-transfer-language fault model. A paper [31] presents the results of several experiments conducted using a fault injection-based automated testing system where the experiments consisted of exhaustively injecting three separate fault types into various locations of two distinct applications. Fault injection has been used to evaluate the dependability of computer systems, but most fault-injection studies concentrate on the final impact of faults on the system with an emphasis on fault latency and coverage issues, but a paper [32] develops a fault injection and monitoring environment as a tool to study fault propagation.

III. ALTERNATE APPROACHES

There are three arguments that we do not need to perform an experiment. The first is that enough flights have been completed to establish the hardware is reliable. The second is that we can establish reliability by a safety case. The third is that we can compute the reliability using a semi-

Markov model of the system. We consider each, describe their limitations, and observe what each has to contribute to conducting a laboratory experiment.

A. Natural Life Testing

A natural life experiment would subject an embedded controller to a certain number of operating periods where we base the number of periods on the desired probability of failure and desired confidence level. The motivation for the confidence level is that if there are random elements present an experiment can mislead us. We take the interpretation of confidence level as the complement of the probability that the experiment has misled us. A confidence level of 99% means there is a 1% chance (or less) that the experiment has misled us. Since a confidence level is a quantitative measure of the quality of the experiment, we will ask that the quality of the experiment match the quantity being measured. For a probability of failure $\leq$ 1e-9, we require a confidence level of 100(1 – 1e-9)% .

A natural-life experiment consists of binary trials where the outcome of any trial is success or failure. If no failures are observed, the formula for the number of trials required, n, to establish a probability of failure $\leq$ p at a confidence level of 100(1 – γ)% is given by the formula

$$(1-p)^n = \gamma \tag{1}$$

For p = γ = 1e-9, n = 21 billion.

The size of the US commercial fleet is about 7,000, and we have had fly-by-wire for about 30 years [51, 52, 53]. Assume every aircraft has the same flight control computer, aircraft fly continuously, and there are 10,000 hours in a year. To establish the probability of failure during a ten-hour flight is $\leq$ 1e-9 at the 100(1 – 1e-9)% confidence level requires the 7,000 aircraft to fly 3,000 years with no failures.

The experiment will simulate the 21 billion trials. As mentioned, we will gain efficiency by using the structural argument that the system will operate correctly if fault free. We will retain realism by using a fault occurrence model based on the failure rate of components. If the failure rate is low, the model will tell us that many operating periods have no fault occurrences.

B. Safety Cases

There are two methods used in safety cases: prescriptive and performance based [38]. Currently, safety cases principally rely on prescriptive methods that set guidelines for product features and development processes. Required product features may include elements such as fail-safe or back-up systems. Standards for development processes specify the procedure used in producing the product. In contrast, performance based criteria consider desired, measurable outcomes, leaving how to achieve the criteria to the designer. There is a general desire for the program to use more performance-based standards, especially among vendors supplying products for approval. This paper develops a performance-based procedure.

On the positive side, we assume a safety case has established at an acceptable level that the system operates correctly if the components are fault free.

C. Semi-Markov Models

An approach to verification is using a semi-Markov model to compute the probability of failure. A problem with this approach is that we need an extremely accurate description of the reconfiguration process to be authentic. The problem is heightened because there are multiple parameters which increases the experimental burden to maintain an overall confidence level, we must increase the confidence level for each parameter. The general result is that the loss of confidence is additive. We will also use the theorem below in the next section when we consider placing components on test.

Theorem [39]: Suppose $[\alpha_i, \beta_i]$ is a $100(1 - h_i)\%$ confidence interval for p_i for $1 \leq i \leq n$, then $([\alpha_1, \beta_1],..., [\alpha_n, \beta_n])$ is a $100(1- h_1 - ... - h_n)\%$ confidence interval for the parameters $(p_1,..., p_n)$.

Given a reliability requirement of 1e-9 and four parameters, each parameter must be estimated at the $100(1 – 2.5e\text{-}10)\%$ confidence level. Describing system reconfiguration at this confidence level appears intractable since reconfiguration is a complex procedure consisting of fault detection, fault identification, and faulty component removal that involves both hardware and software.

We illustrate this with a simple model, depicted in figure 1, of a reconfigurable fourplex subject to permanent faults. In this model, λ is the failure rate of a component. D_1 and D_2 are the diagnostic level for the system as a fourplex and as a threeplex. The difficult parameters to estimate are the reconfiguration transition functions δ_1 and δ_2 that remove component identified as faulty from the system. S represents the states that are fault free; R the recovery mode states; F the failure states due to undetected faults; C the failure states due to nearly coincident faults; and E the failure state due to exhaustion of parts.

Instead of attempting to determine the reconfiguration density function at an ultra-high level, a modeler can attempt to decompose it into several simpler steps. An example is in figure 2 where the three steps are detection, identification, and removal with constant rate transitions.

One objection to this decomposition is that some elements of reconfiguration are fixed-time procedures that are not modeled by constant rate (memoryless) transitions. Another is that the intermediate states of detection and identification may not be observable, making it difficult to estimate the transitions between them.

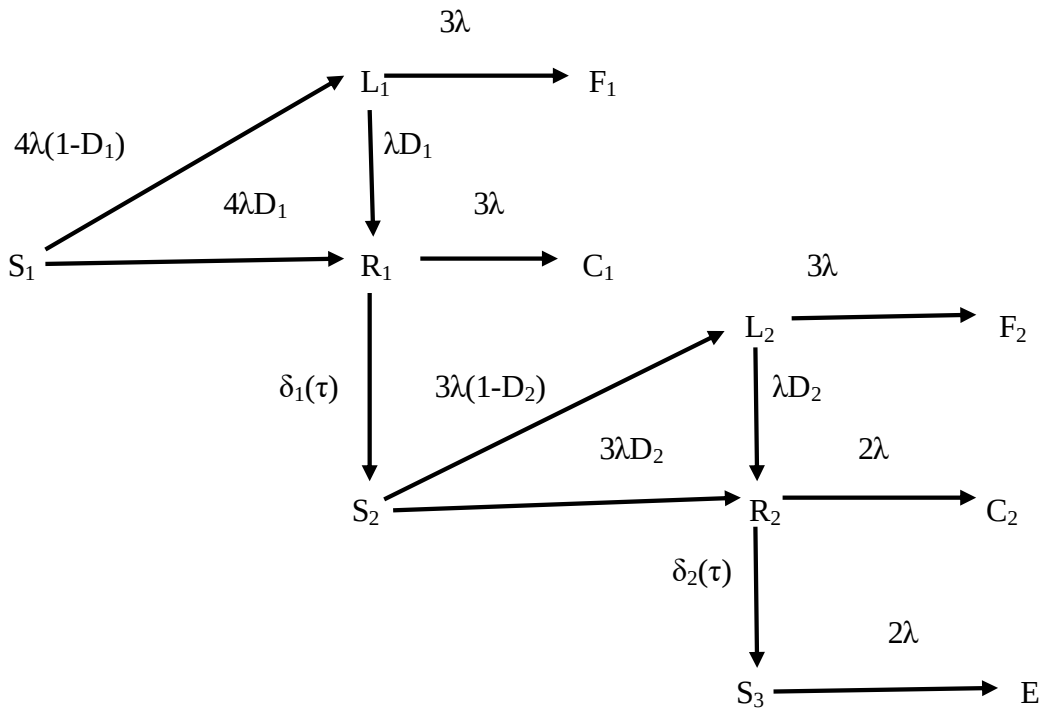

Figure 1: Semi-Markov model of a reconfigurable fourplex with only permanent faults

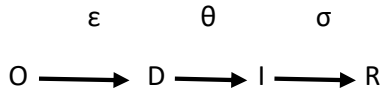

Figure 2: Proposed decomposition of the reconfiguration process

In general, no detail of reconfiguration can be ignored since any ignored element can have a greater effect on the computation than the extremely small quantity to be computed, but a detailed description of reconfiguration yields intractable models with hidden states.

This subsection does not claim that establishing ultra-reliability by semi-Markov models is impossible. It merely points out that the current approach appears intractable.

IV. Nature of Faults

We consider the definition of a fault used in this paper, an alternate definition, and establishing the failure rate of commercial off the shelf components.

A. Definition of Fault for this Paper

In this paper, a fault is an input-output malfunction of a device. That lets a fault be observable when collecting field data and increases the likelihood of detection when it occurs in a system although there are obstacles to both objectives. One is that we may have the failure rate of a component but we do not know the faulty behavior of a component. To counter this, design a system to detect any input-output malfunction.

Another is that observability requires integrated circuits to be practical. Consider the simplified computer in figure 3 where the chip shaded black has become faulty. In the depicted configuration, it has no direct link to the output pins, and consequently, it's not certain we can get its faulty behavior to the computer's output pins.

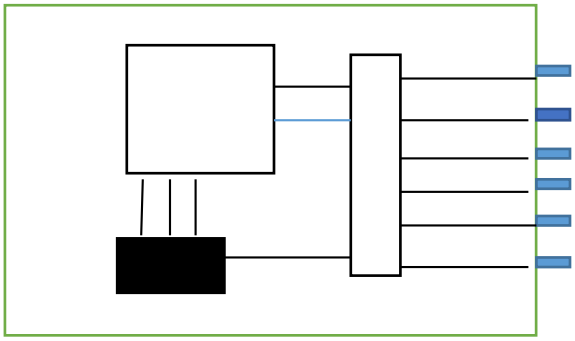

Figure 3: Computer with moderate level of circuit integration

Most likely, we would have to redesign the computer to achieve a high diagnostic level. The new design would not be as efficient, but it would be a question of effectiveness (verifiability) being more important.

For this reason, for this experiment, we assume we have a computer on a chip, and its failure rate refers to the moment when incorrect values begin to appear on its output pins.

For future consideration, one method of achieving a high diagnostic level is to have dual units that flag an error if they disagree. For four computers, an architecture would be a double-dual where each pair acts in lockstep.

B. Alternate Definition of Faults

Another definition is that a fault is a defect that may or may not cause an error while an error is an observable manifestation of a fault [50]. By this set of definitions, the experiment in this paper would be called error injection. The definition of a fault as a defect that may or may not cause an error increases the likelihood that a fault will not be detected.

Undetected faults receive attention because of the possibility they will accumulate until some event causes them to produce multiple errors that overwhelm the system. If faults are accumulating in components that can eventually cause the system to fail, then the components are wearing out. For components that wear out, there is often a replace-before-failure policy based on the wear-out distribution and the age of the component. In this case, we would derive the wear-out distribution from the percentage of faults that are not detected.

C. Commercial-Off-The-Shelf (COTS) Components

Commercial off the shelf components offer an opportunity to establish their failure rate because they are inexpensive and because a constant failure rate lets us trade numbers for time. We consider a confidence level higher than the reliability since a system can use different types of components. For the computations below, we assume a maximum of ten different components. To establish a probability of failure $\leq$ 1e-4/hour at the 100(1 – 1e-5)% level requires placing 54 components on test for three months and observing no failures. For 1e-5/hour at the 100(1 – 1e-6)% level, place 320 components on test for six months.

In contrast, to establish probability of failure $\leq$ 1e-6/hour at the 100(1 – 1e-7)% level requires placing 1000 units on test for 2 years.

Trading numbers for time permits conducting the test in a reasonable amount of time, but it may not let us determine whether or not the components are wearing out.

D. Relevant Literature

A paper considers imperfect diagnostics and computes the conditional probability of system failure during an operating period given the system has not yet failed [33]. A source for the failure rate of components is [34]. Discussions of the reliability of COTS are in [35, 36, 37].

V. EXPERIMENTAL APPROACH

We wish to demonstrate fault tolerance, and we base the experiment on modified natural-life testing for a class of systems. The systems are those that have an operating period with no maintenance followed by maintenance between operating periods. A natural life test would subject the system to a number of operating periods, and if we observe no failures, declare the system has a certain reliability at some confidence level. The number of trials is given by formula (1).

In a simulated natural-life test, each trial simulates an operating period, but it gains efficiency by using the structural argument that the system operates correctly if the components are fault free.

Using this result, we need only observe the system from fault injection to system recovery. In a well-designed system, recovery is fast in order to prevent an accumulation of faults that could overwhelm the system.

Using the failure rate of the components, we develop a model that describes all possible fault occurrences and derive theorems that tell us how many, what kind, when and where to inject faults during a trial.

The simulated life test described in this paper does impose failure conditions that are not present during natural life testing. In a natural life test, the system need only survive the required number of operating periods. It does not need to detect the faults or correctly identify the faults or reconfigure correctly. We require all of these in the simulated test. Furthermore, the system needs to perform these actions quickly to achieve the necessary efficiency.

In a complete test, during fault recovery, the system must maintain process control or plant monitoring. If the system does not perform its intended function during fault recovery, that counts as a failure.

Finally, any experiment with experimental error and a confidence level requires that the system be more reliable than originally required to pass the test. For example, suppose the requirement is a probability of failure $\leq$ 1e-9. If the system has a probability of failure of 1e-9 and we conduct 21 billion trials, we can expect 21 failures during the experiment.

VI. The Global Fault Model

This section constructs a global fault model and derives theorems about the model that show how to choose the number, occurrence time, type, and location of the faults for a trial (representing one operating period) in the experiment. The model assumes the events are independent and occur at a constant rate, but the events are general. The event could be a permanent, transient, or intermittent fault. It could be independent faults or faults correlated in time or space or both. It could represent one fault inducing other faults. The first four subsections review four standard distributions. The fifth derives a basic result on fault occurrence and the sixth interprets the result in terms of the four standard distributions.

A. Independent and Competing Constant Rate Events

Suppose there are m events each with rate α_i as depicted in figure 2. In the exposition below these events will be a detailed description of all possible fault occurrences. For instance, S_1 can be the occurrence of a permanent fault in component 1, while S_2 can be the occurrence of a transient fault in component 1.

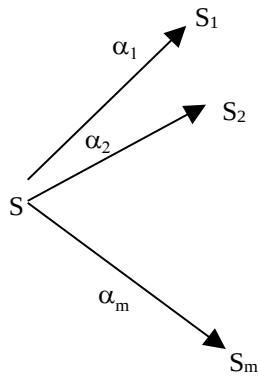

Figure 4: The Fan-out for fault occurrence

For the model in figure 4, the probability that event i has occurred given some event has occurred is

$$\frac{\alpha_i}{\alpha_1 + \ldots + \alpha_m} \tag{2}$$

B. The Poisson Distribution

The Poisson distribution is a renewal process that occurs at a constant rate. The model with rate β is given in figure 5.

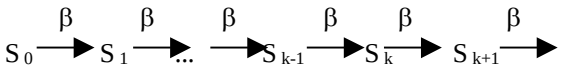

Figure 5: Poisson Renewal Process

For the model in figure 3, the probability of being in state k at time T is

$$\frac{(\beta T)^k e^{-\beta T}}{k!}. \tag{3}$$

The fault injection procedure will have to be adapted to the assumption of a constant rate used by the Poisson process. If the system removes failed components, the failure rate of the system does not remain constant. One method of handling this is to treat the removed components as virtual components. This means that the component is theoretically subject to later fault injections, but in practice these faults will not be injected if the system has already removed the component. If the system has not yet removed the faulty component, then the second fault can be injected into the same component. This double injection checks that the occurrence of a second fault does not interfere with the detection and removal of a faulty component.

C The Ordered Uniform Distribution

Choose a sample of size n $(x_1, x_2, \ldots, x_n)$ from the uniform distribution on the interval [0 T]. Order it as $x_{(1)} \le x_{(2)} \le \ldots \le x_{(n)}$. The distribution

$$\text{Prob}\{x_{(1)} \le s_1, x_{(2)} \le s_2, \ldots, x_{(n)} \le s_n\} \tag{4}$$

$$= \frac{n!}{T^n} \int_0^{s_1} \int_{t_1}^{s_2} \cdots \int_{t_{n-1}}^{s_n} dt_n \cdots dt_2\, dt_1$$

is called the ordered uniform distribution.

D. The Multinomial Distribution

Suppose we sample with replacement from a population with m classes of objects. Suppose the probability of choosing an object from class i is p_i. For a sample of size n the probability of choosing k_i objects from class i (i=1,...,n) is

$$\frac{n!}{k_1!\, k_2! \cdots k_m!} p_1^{k_1} p_2^{k_2} \cdots p_m^{k_m} \tag{5}$$

In particular, if the class of objects is the set of faults given in figure 4, then the probability of k_i faults of type i occurring given n faults have occurred is given by the expression

$$\frac{n!}{k_1!\, k_2! \cdots k_m!} \left(\frac{\alpha_1}{\alpha}\right)^{k_1} \left(\frac{\alpha_2}{\alpha}\right)^{k_2} \cdots \left(\frac{\alpha_m}{\alpha}\right)^{k_m} \tag{6}$$

where $\alpha = \alpha_1 + \cdots + \alpha_m$. In the formulas (5) and (6) some of the k_i's can be zero.

E. Global fault model

Suppose there are m classes of faults with α_i the rate for class i. The total rate is $\alpha = \alpha_1 + \ldots + \alpha_m$. The global model for precisely n faults occurring during an operating period is given in figure 4. The fan out from all the intermediate states is the same as the fan out from state S_0. There are n+2 columns beginning with column 0 and ending with column n+1. The first and last columns have a single row. The process begins in state S_0 and ends in one of the states in the n^{th} column. The final state S_{n+1} is included because the mathematical formulation specifies that precisely n faults have occurred (during the operating period) by requiring that the process reaches the n^{th} column during the operating period, but the transition into S_{n+1} does not occur until after the operating period.

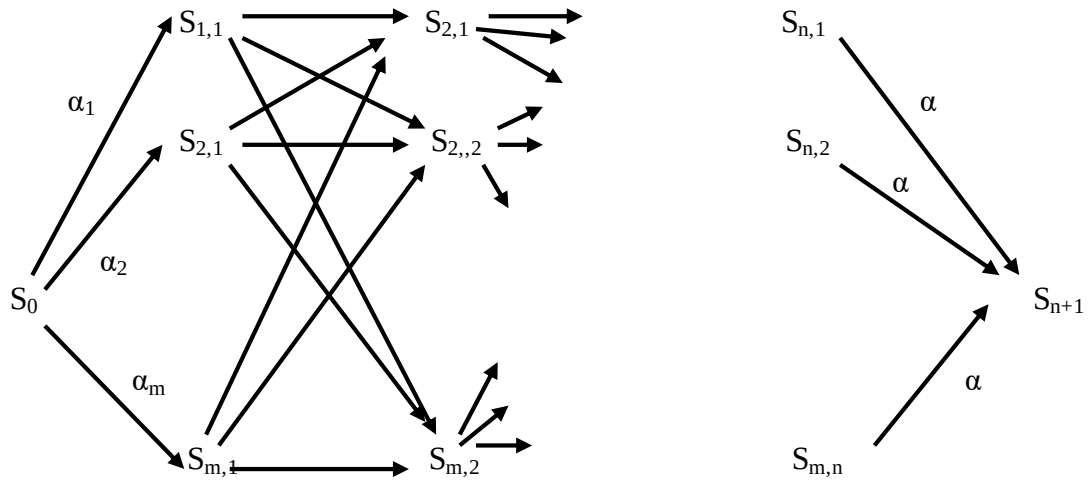

Figure 6: Global fault occurrence model

Suppose k_i faults of type i (i = 1,..., m) with $n = k_1 + ... + k_m$) have occurred in some specified order, and let β_j be the rate of the j^{th} fault (j=1,..., n) that has occurred. We have,

$$\beta_1 \beta_2 \cdots \beta_n = (\alpha_1)^{k1} (\alpha_2)^{k2} \cdots (\alpha_m)^{km} \qquad n = k_1 + k_2 + \ldots + k_m \tag{7}$$

and $\alpha = \alpha_1 + ... + \alpha_m$. The probability that the n designated faults occur in the designated order and that the j^{th} fault occurs before time s_j is given by the convolution integrals below. The expression in brackets is the probability that the transition into state S_{n+1} does not occur, ensuring that precisely n faults have occurred

$$\int_0^{S_1} \beta_1 e^{\beta_1 t_1} e^{-(\alpha-\beta_1) t_1}$$

$$\int_{t_1}^{S_2} \beta_2 e^{\beta_2 (t_2 - t_1)} e^{-(\alpha-\beta_2)(t_2 - t_1)}$$

$$\vdots$$

$$\int_{t_{n-1}}^{S_n} \beta_n e^{\beta_n (t_n - t_{n-1})} e^{-(\alpha-\beta_n)(t_n - t_{n-1})}$$

$$\left[1 - \int_{t_n}^{T} \propto e^{-\propto (t_{n+1} - t_n)} dt_{n+1}\right]$$

$$dt_n \ldots dt_2 \, dt_1$$

$$= \beta_1 \beta_2 \ldots \beta_n e^{-\propto T}$$

$$\int_0^{S_1} \int_{t_1}^{S_2} \ldots \int_{t_{n-1}}^{S_n} dt_n \ldots dt_2 \, dt_1$$

$$= (\propto_1)^{k_1} \dots (\propto_m)^{k_m} e^{-\propto T}$$

$$\int_0^{s_1} \int_{t_1}^{s_2} \dots \int_{t_{n-1}}^{s_n} dt_n \dots dt_2\, dt_1 \qquad (8)$$

This last expression is the occurrence probability for precisely n faults of a specified type and location in a specified order at specified times.

Expression (8) does not depend on the specified order of the faults. Since any ordering yields the same probability, all orderings are equally likely. Hence the occurrence probability for precisely n faults of specified type and location at specified times in any order whatsoever is

$$\frac{n!}{k_1!\, k_2! \cdots k_m!} \left(\alpha_1\right)^{k1} \left(\alpha_2\right)^{k2} \dots \left(\alpha_m\right)^{km} e^{-\alpha T} \qquad (9)$$

$$\int_0^{s_1} \int_{t_1}^{s_2} \dots \int_{t_{n-1}}^{s_n} dt_n \dots dt_2\, dt_1$$

F. Interpretation of the derivation

We have

$$\frac{n!}{k_1!\ k_2! \dots k_n!} (\propto_1)^{k_1} \dots (\propto_m)^{k_m} e^{-\propto T} \int_0^{s_1} \int_{t_1}^{s_2} \dots \int_{t_{n-1}}^{s_n} dt_n \dots dt_2\, dt_1$$

$$= \frac{n!}{k_1!\ k_2! \dots k_n!} (\propto_1)^{k_1} \dots (\propto_m)^{k_m} e^{-\propto T}$$

$$\left[\frac{\propto^n}{\propto^n} \frac{n!}{n!} \frac{T^n}{T^n}\right] \int_0^{s_1} \int_{t_1}^{s_2} \dots \int_{t_{n-1}}^{s_n} dt_n \dots dt_2\, dt_1$$

$$= \left[\frac{n!}{k_1!\ k_2! \dots k_n!} \left(\frac{\propto_1}{\propto}\right)^{k_1} \dots \left(\frac{\propto_m}{\propto}\right)^{k_m}\right] \left[\frac{(\propto T)^n e^{-\propto T}}{n!}\right]$$

$$\left[\frac{n!}{T^n} \int_0^{s_1} \int_{t_1}^{s_2} \dots \int_{t_{n-1}}^{s_n} dt_n \dots dt_2\, dt_1\right] \qquad (10)$$

The expression (9) (and hence the probability) is algebraically equivalent to a product of three probability distributions as given in expression (10). The multiplicative property implies these three distributions act independently.

The three distributions are the Poisson renewal process, the ordered uniform, and the multinomial distribution. Since they act independently, the faults for any trial (representing one operating period) in the simulation can be chosen as follows.

Theorem:

(i)The number of faults is given by the Poisson.

(ii)The occurrence times are given by the ordered uniform

(iii)At each occurrence time the location and type of fault is given by the multinomial

(iv)These distributions act independently

We might think that since transients occur at a faster rate than permanents that if a transient and a permanent have occurred then it is more likely that the transient has occurred first, but both orderings are equally likely. Another theorem about the global fault model is the following.

Theorem: Given that a set of faults has occurred, all orderings are equally likely.

Even though all orderings are equally likely, different orderings can have different effects on the system. Suppose we have a fourplex, and a transient and permanent occur. If the permanent occurs first, then the permanent encounters a fourplex while the transient encounters a threeplex. If the transient occurs first and there is sufficient time between the occurrences for the system to have recovered from the transient, both the transient and the permanent encounter a fourplex.

G. Partitioning to Improve Efficiency

This subsection discusses using knowledge about system structure and characteristics to improve the efficiency of the experiment. As described above, for each trial the number of injected faults is chosen by random sampling. Likewise, the time and place for fault injection.

That the number of trials with k fault occurrences is a random variable introduces several problems. First, it is possible to randomly choose a large number of faults for a single trial. For instance, for the fourplex example any trial with three or more faults has the potential for causing the system to fail. Second, it can also produce a large number of trials with no or few faults. Both of these events can cause concern about the entire experiment being misleading because of an unusual run of random numbers.

The procedure divides the trials into three classes according to the fault occurrences during the trial. The first class, labeled BF for benign faults, are those trials that have fault occurrences that will not cause system failure. The second class, labeled IF for injected faults, are those to be studied by experiment. The third class, labeled CF for catastrophic faults, are those trials we declare to cause system failure because of the fault occurrences during the trial.

The next part requires careful exposition since we need two different quantities. One is an upper bound on P{fail | IF}, the probability of system failure given the set of injected faults. The other is a confidence level (the complement of the probability that the experiment has misled us) for the upper bound on P{fail | IF}.

Suppose BF is the set of benign faults that do not cause system failure, CF is the set of catastrophic faults assume to cause system failure, and IF the set of faults we will inject. Then

$$P\{Fail\} = P\{Fail \mid BF\}P\{BF\} + P\{Fail \mid IF\}P\{IF\} + P\{Fail \mid CF\}P\{CF\}$$

$$= 0 + P\{Fail \mid IF\}P\{IF\} + P\{CF\} \quad (11)$$

or

$$P\{Fail \mid IF\} = [\, P\{Fail\} - P\{CF\} \,] / P\{IF\}. \quad (12)$$

Using the frequentist interpretation that the confidence level is the complement of the probability that the experiment has misled us, P{misled}, gives

$$P\{misled\} = P\{misled \mid BF\}P\{BF\} + P\{misled \mid IF\}P\{IF\} + P\{misled \mid CF\}P\{CF\} \quad (13)$$

Now, P{misled | BF} = 0. For P{misled | CF}, we reason that we are considering a one-sided confidence level: the complement of the probability that the experiment has misled us into thinking that the system is reliable when it is not. Declaring the system will fail if certain faults have occurred has zero chance of misleading us that the system is reliable. Hence, P{misled | CF} = 0. We have

$$P\{misled \mid IF\} = P\{misled\} \; / \; P\{IF\}. \tag{14}$$

Given no failures are observed, the number of trials needed to establish the required confidence level is given by using formula (1).

$$n = [\, \log(\, P\{misled \mid IF\}\,] \; / \; [\, \log(\, 1 - P\{Fail \mid IF\}\,)\,]. \tag{15}$$

VII. FOURPLEX ARCHITECTURE

This section describes a reconfigurable fourplex and its diagnostic routine.

A. System Structure and Fault Classes

The system has four processors and twelve unidirectional links depicted in figure 7.

We consider both permanent and transient faults. A transient lasts one control cycle. A faulty processor sends three (different) random numbers to the other processors. A faulty link sends a random number to its destination processor. We assume a 16-bit machine and choose the random numbers from the uniform distribution on 1 to 65536.

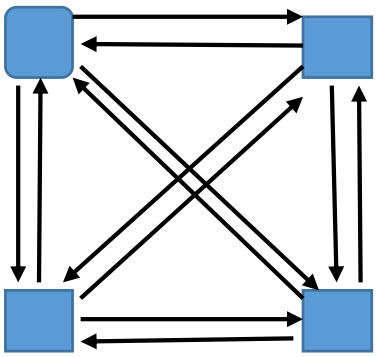

Figure 7: Reconfigurable Fourplex

B. Diagnostic Procedure

The diagnostic routine is a three-step procedure performed during a single control cycle; it uses each processor's control command; and the system performs the diagnostic procedure before sending its majority-determined control command to the actuators.

In the first step, each processor sends its result to the other processors. Each processor compares what it receives from the other processors to its result. It assigns a 1 to another processor if it agrees with the other processor and a 0 if it does not. A processor always agrees with its own result. Each processor forms a vector of zeros and ones.

$$\begin{bmatrix} comparison\ with\ processor\ 1 \\ comparison\ with\ processor\ 2 \\ comparison\ with\ processor\ 3 \\ comparison\ with\ processor\ 4 \end{bmatrix}$$

In the second step, each processor sends its vector to the other processors. Each processor now has a 4x4 matrix with column j containing the vector it received from processor j. Each processor examines its 4x4 matrix and decides which, if any, component is faulty.

In the third step, each processor sends its diagnostic to the other processors. All processors vote, and a majority of good processors record their vote and send their control command to the actuators.

At the completion of the fifth control cycle, each processor considers its five diagnostic results, performs a three-out-of-five vote to determine if a component is permanently faulty, and sends its results to the other processors. If a majority decide a component is permanently faulty, it is removed from the system.

If processor k is faulty, the good processors will have four zeros in row k. If the link from j to k is faulty, the processors other than k will have a zero in row j and column k.

C. Examples

Suppose processor 1 is faulty. After completion of the second step above,

processor 1 has the matrix $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$

while processors 2, 3 and 4 have the matrix $\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}$

By examining the first row, processors 2, 3, and 4 decide that processor 1 is faulty during that control cycle.

Suppose the link from processor 1 to processor 2 is faulty. After completion of the second step,

processors 1, 3, and 4 have the matrix $\begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$

while processor 2 has the matrix $\begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}$

Examining the matrices, all processors decide that the link from processor 1 to processor 2 is faulty during that control cycle. It is not a faulty processor since no row has three or more zeros.

This diagnostic procedure only handles a single fault. The next section will show that if the diagnostic procedure always correctly identifies the faulty component within five control cycles during the experiment, then the diagnostic procedure does not need to handle two faulty components for the system to be highly reliable.

D. Reduced Fourplex

If a processor is removed, the remaining processors consider a 3x3 matrix. If a link is removed, the processors consider a 4x4 matrix where the removed link is assumed to have sent a 1. Since the diagnostics consider the pattern of zeros, the removed link is ignored.

D. Lack of Diagnostics

There are two sources for the lack of diagnostics. First, some diagnostics will have to be performed during maintenance. If a processor fails, the system ignores the six links connecting that processor to the others. If two links from a processor fails, the system ignores that processor, its third link, and the three links to that processor. Second, the random numbers can match the correct answer. This is highly unlikely for the three-out-of-five vote for a permanent fault, but there is a 1 in 65536 chance for a transient link fault. Missing a transient fault, however, does no harm.

VIII. Designing an Experiment for the Fourplex

A. Overview

This section applies the theorems of section six to the reconfigurable fourplex described in the previous section and depicted in figure 7. This system permits applying all the results in section six.

There are two failure conditions for this example (i) if the system is operating as a twoplex and a fault occurs, or (ii) if the system does not detect and correctly identify a fault within five control cycles. The second condition is modified for transient faults: It is acceptable to ignore a transient fault, but it is a failure if the system incorrectly identifies the fault. This initial example does not monitor performance.

The goal of this section is to determine what fault injections we need to perform for the trials in the experiment. It uses the theorems on partitioning in section VI to increase the efficiency of the

experiment. Subsection B presents the initial results without regard to partitioning. Subsection C describes the partitioning. Subsections D and E perform the computations for two of the partitions. Subsection F summarizes the results of subsections D and E. Subsection G describes the class of injectable faults and uses its results and the results of subsections C through F to determine the number of trials. Subsection H considers the order of fault occurrence for the injectable faults. The order determines which faults will be injected into a fourplex and which into a threeplex. Subsection I descries the simulation.

B. Initial Numerical Results

The operating time is 10 hours, and the reliability requirement for the system is ≤ 1e-9 chance of failure during the operating period. The control cycle is 50 milliseconds or 720,000 cycles per operating period.

The failure rates for each component are

permanent 1e-6/hour

transient 1e-5/hour

We normalize operating time to 1, which gives component failure rates of

permanent 1e-5/10 hours

transient 1e-4/10 hours

for an overall failure rate of 1.76e-3/operating period. By equation (1), the initial number of trials in round numbers, is n = 21e+9 where each trial simulates one operating period. The probability of a certain number of faults during an operating period and the expected number of trials for that number of faults in given by table 1.

Table 1: Initial numerical results

Number of Faults	Probability	Expected Fault Injections
0	9.9824e-1	2.0687e+10
1	1.7569e-3	3.6409e+7
2	1.5461e-6	3.2040e+4
3	9.0703e-10	1.8796e+1
≥ 4	3.9923e-13	8.3838e-3

The expected injections per number of faults is only an expectation. The actual number is decided by random sampling. In addition, the expected number is only an initial indicator since we are going to modify the experiment by partitioning the sample space as described in section six.

C. Partitioning the Sample Space

As before, BF is the set of benign faults that do not cause system failure, CF is the set of catastrophic faults assumed to cause system failure, and IF the set of faults we will inject.

We divide CF into three parts. CF1 is the large number of faults class. CF2 is nearly coincident faults when there are two fault occurrence during a trial. CF3 is a combination of two phenomenon: nearly coincident faults when there are three fault occurrences during a trial and more than one permanent fault when there are three fault occurrences during a trial.

We base the exclusion criterion for nearly coincident faults on the assumption that the system detects all faults within five control cycles, but this is not guaranteed. If, during the experiment, the system does not correctly identify all faults, then we will have to perform a different analysis.

Two of the classes of faults are immediate:

The class consists of zero fault occurrence: P{BF} = 9.9824e-1.

The large number of fault class is P{CF1} = P{≥ 4 faults} = 3.9923e-13.

The classes CF2 and CF3 ae more complex

D. Computation of CF2

Turning to nearly coincident faults and setting the operating period equal to 1, five control cycles equals Δ = 5/720000. Using the ordered uniform distribution, the probability of nearly coincident faults given two faults is

$$Q1 = 2\int_0^{1-\Delta} 1 \int_{t_1}^{t_1+\Delta} 1 + 2\int_{1-\Delta}^{1} 1 \int_{t_1}^{1} 1 = 2\,\Delta\,(1-\Delta) = 1.3889e-5. \quad (16)$$

Hence, CF2 = P{ two faults } Q1 = 2.1474e-11.

E. Computation of CF3

For nearly coincident faults given three faults, let A be the first and second faults are within Δ, and B be the second and third faults are within Δ. P{A or B} = P{A} + P{B} - P{A and B}.

$$P\{A\} = 6\int_0^{1-\Delta} 1 \int_{t_1}^{t_1+\Delta} 1 \int_{t_2}^{1} 1 + 6\int_{1-\Delta}^{1} 1 \int_{t_1}^{1} 1 \int_{t_2}^{1} 1 = 3\Delta - 3\Delta^2 + \Delta^3 \quad (17)$$

$$P\{B\} = 6\int_0^{1} 1 \int_{t_1}^{1-\Delta} 1 \int_{t_2}^{t_2+\Delta} 1 + 6\int_0^{1} 1 \int_{1-\Delta}^{1} 1 \int_{t_2}^{1} 1 = 3\Delta - 3\Delta^2 \quad (18)$$

$$\begin{aligned} P\{A \text{ and } B\} = {} & 6\int_0^{1-2\Delta} 1 \int_{t_1}^{t_1+\Delta} 1 \int_{t_2}^{t_2+\Delta} 1 \\ & + 6\int_{1-2\Delta}^{1-\Delta} 1 \int_{t_1}^{1-\Delta} 1 \int_{t_2}^{t_2+\Delta} 1 + 6\int_{1-2\Delta}^{1-\Delta} 1 \int_{1-\Delta}^{t_1+\Delta} 1 \int_{t_2}^{1} 1 \\ & + 6\int_{1-\Delta}^{1} 1 \int_{t_1}^{1} 1 \int_{t_2}^{1} 1 \\ = {} & 6\Delta^2 - 6\Delta^3 \end{aligned} \quad (19)$$

Hence, the probability of a nearly coincident fault given three faults is Q2 = $P\{A\} + P\{B\} - P\{A \; and \; B\} = 6\Delta - 12\Delta^2 + 7\Delta^3$ = 4.1666e-5

If three faults occur, three permanents or two permanents followed by a transient could cause the system to fail. We restrict the three-fault case to zero or one permanent. The probability a fault is permanent is p = 1/11. The probability of more than one permanent given three faults is

$$Q3 = \; p^3 + 3\,p^2\,(1-p) = 2.3291e-2 \tag{20}$$

We combine the above to get the probability of excluded faults for three fault occurrences.

$$P\{CF3\} = P\{\text{ 3 faults }\}\,[\,Q2 + Q3 - Q2Q3] = 2.1163e\text{-}11. \tag{21}$$

F. Computation for the class CF

Since P{CF1}, P{CF2}, and P[CF3} are probabilities of disjoint sets,

$$P\{CF\} = P\{CF1\} + P\{CF2\} + P\{CF3\} = 4.3036e\text{-}11. \tag{22}$$

G. The class IF

Turning to the injected faults, let IF1 be the occurrence of a single fault; IF2 be the occurrence of two faults separated by Δ; and IF3 be the occurrence of three faults containing nor more than one permanent and separated by Δ.

$$P\{IF1\} = P\{\text{single fault}\} = 1.7569e\text{-}3 \tag{23}$$

$$P\{IF2\} = P\{\text{two faults}\} \times [1-Q1] = 1.5461e\text{-}6 \tag{24}$$

$$P\{IF3\} = P\{\text{three faults}\} \times [1 - Q2 - Q3 + Q2Q3] = 8.8587e\text{-}10 \tag{25}$$

Since P{IF1}, P{IF2}, and P{IF3} are probabilities of disjoint sets,

$$P\{IF\} = P\{IF1\} + P\{IF2\} + P\{IF3\} = 1.7584\ e\text{-}3 \tag{26}$$

By formula (12), the required upper bound on P{Fail | IF} is

$$P\{Fail \mid IF\} = [\,P\{Fail\} - P\{CF\}\,] \,/\, P\{IF\} = 5.4422e\text{-}7 \tag{27}$$

By formula (14), the required confidence level is

$$P\{misled \mid IF\} = P\{misled\} \,/\, P\{IF\} = 5.6870e\text{-}7 \tag{28}$$

Given no failures are observed, the number of trials needed to establish the required confidence level is given by substituting into formula (15).

$$n = [\,\log(\,P\{misled \mid IF\}\,]\,/\,[\,\log(\,1 - P\{Fail \mid IF\}\,)\,] = 2.6423e+7 \tag{29}$$

H. The Order of Fault Occurrence for the Class IF

We consider the order of fault occurrence since that determines whether we inject a fault into a threeplex or a fourplex For instance, suppose two faults occur: a permanent and a transient. If the

first fault is permanent, the second fault is injected into a threeplex whereas if the first fault is transient, the second fault is injected into a fourplex.

Suppose r = probability a fault is permanent fault = 1/11

$$P\{\text{perm-perm} \mid \text{two faults}\} = r^2 = 8.2645\text{e-}3 \quad (30)$$

$$P\{\text{perm-tran} \mid \text{two faults}\} = r(1-r) = 8.2645\text{e-}2 \quad (31)$$

$$P\{\text{tran-perm} \mid \text{two faults}\} = r(1-r) = 8.2645\text{e-}2 \quad (32)$$

$$\text{Prob}\{\text{tran-tran} \mid \text{two faults}\} = (1-r)(1-r) = 8.2645\text{e-}1 \quad (33)$$

and

$$P\{\text{perm-tran-tran} \mid 3 \text{ faults}\} = r(1-r)(1-r) = 7.5131\text{e-}2 \quad (34)$$

$$P\{\text{tran-perm-tran} \mid 3 \text{ faults}\} = r(1-r)(1-r) = 7.5232\text{e-}2 \quad (35)$$

$$P\{\text{tran-tran-perm} \mid 3 \text{ faults}\} = r(1-r)(1-r) = 7.5131\text{e-}2 \quad (36)$$

$$P\{\text{tran-tran-tran} \mid 3 \text{ faults}\} = (1-r)(1-r)(1-r) = 7.5131\text{e-}1 \quad (37)$$

Using the result that the number of faults and the type of faults act multiplicatively, we have

$$P\{IF1\} = 1.7569\text{e-}3 \quad (38)$$

$$P\{\text{IF2 and p-p}\} = P\{IF2\}\, P\{\text{perm-perm} \mid \text{two faults}\} = 1.2778\text{e-}8 \quad (39)$$

$$P\{\text{IF2 and p-t}\} = P\{IF2\}\, P\{\text{perm-tran} \mid \text{two faults}\} = 1.2778\text{e-}7 \quad (40)$$

$$P\{\text{IF2 and t-p}\} = P\{IF2\}\, P\{\text{tran-perm} \mid \text{two faults}\} = 1.2778\text{e-}7 \quad (41$$

$$P\{\text{IF2 and t–t}\} = P\{IF2\}\, \text{Prob}\{\text{tran-tran} \mid \text{two faults}\} = 1.2778\text{e-}6 \quad (42)$$

$$P\{\text{IF3 and p-t-t}\} = P\{IF3\}\, \text{Prob}\{\text{tran-tran} \mid \text{two faults}\} = 6.6556\text{e-}11 \quad (43)$$

$$P\{\text{IF3 and t-p-t}\} = P\{IF3\}\, P\{\text{tran-perm-tran} \mid 3 \text{ faults}\} = 6.6556\text{e-}11 \quad (44)$$

$$P\{\text{IF3 and t-t-p}\} = P\{IF3\}\, P\{\text{tran-tran-perm} \mid 3 \text{ faults}\} = 6.6556\text{e-}11 \quad (45)$$

$$P\{\text{IF3 and t-t-t}\} = P\{IF3\}\, P\{\text{tran-tran-tran} \mid 3 \text{ faults}\} = 6.6556\text{e-}10 \quad (46)$$

Since we sample from the class IF, we condition the above quantities by P{IF}. The resulting conditional probabilities and the resulting expected number of trials for each sub-partition given the total number of trials is 2.6423e+7 is given by table 2.

Table 2: Number of trials based on partitioning

Conditional Probability	Expected Number of Trials Based on Conditional Probability
P{IF1}/P{IF} = 9.9915e-1	2.6401e+7
P{p-p and IF2}/P{IF} = 7.2668e-6	1.9201e+2
P{p-t and IF2}/P{IF} = 7.2668e-5	1.9201e+3
P{t-p and IF2}/P{IF} = 7.2668e-5	1.9201e+3
P{t-t and IF2}/P{IF} = 7.2668e-4	1.9201e+4
P{p-t-t and IF3}/P{IF} = 3.7850e-8	1.0001
P{t-p-t and IF3}/P{IF} = 3.7850e-8	1.0001
P{t-t-p and IF3}/P{IF} = 3.7850e-8	1.0001
P{t-t-t and IF3}/P{IF} = 3.7850e-7	1.0001e+1

I. Simulation

A simulation checked the diagnostic routine. At the beginning of each trial, it chose the faults according to the distributions described above. The diagnostic routine detected and correctly identified the faults for all the 27 million trials. The exception was that about one in every 100 thousand trials, it did not detect a transient link fault because the random error matched the correct value. Not detecting the transient link fault had no effect on the reliability of the system.

One million trials on a desktop running an interpretive language took 90 seconds. All the trials took less than one hour. For a more complex system, we can run 1000 hours, use 100 desktops, and shift to a compiled language. We can accept a system five or six orders of magnitude more complex.

The fourplex example did not include any applications, but a simulation of a complete system would check that the system maintained process control or plant monitoring during fault detection and recovery. In this case, the time of fault occurrence and the condition of the plant would be a factor. The time of occurrence and the condition of the plant can be chosen randomly from normal operating conditions. In addition, the experimenters may wish to conduct some trials simulating the more hazardous operating conditions. For instance, if a quadcopter is required to stay within a boundary, the experiment can include numerous trials where the quadcopter is close to the boundary.

One method of increasing the likelihood that the system maintains performance during fault recovery is to designate time slots during a control cycle dedicated to fault detection and recovery. With this scheme, it may take longer to recover from a fault than if the system dedicated itself to fault recovery. If faults are infrequent, this is an overhead that will seldom be used. This approach requires a system with higher performance.

IX. Integrating an Argument from Design

We consider integrating an argument-from-design and statistical analysis for the design of the experiment. A result in computer science is that a system of 3k+1 components can correctly identify k faulty components in a system. Hence, while a reconfigurable fourplex has four components, it can correctly identify a single faulty component. There are two cases depending on whether or not the argument-from-design establishes the number of control cycles it takes to identify the faulty component.

A. First Case

In our first example, we assume there is a demonstration that all faults will be correctly identified when there are four components in the system but no demonstration about detection within a time limit although we assume the system detects the fault sometime during the operating period. In this case, we can classify the trials with a single fault as benign. We have

$$P\{BF\} = P\{\ 0 \text{ faults}\} + P\{\ 1 \text{ fault}\} = 9.999969e\text{-}1 \quad (47)$$

The injectable faults are 1F2 and 1F3 of the previous section, which gives

$$P\{IF\} = P\{IF2\} + P\{IF3\} = 1.5470e\text{-}6 \quad 48)$$

The catastrophic faults remain CF = 4.3036e-11

The upper bound for the probability of failure for the injected faults is

$$P\{Fail \mid IF\} = [\ P\{Fail\} - P\{CF\}\ ] / P\{IF\} = 6.1823e\text{-}6 \quad (49)$$

The confidence level for this probability is

$$P\{misled \mid IF\} = P\{misled\} / P\{IF\} = 6.4641e\text{-}4 \quad (50)$$

The number of trials is

$$n = [\ \log(\ P\{misled \mid IF\}\ ] / [\ \log(\ 1 - P\{Fail \mid IF\}\ )\ ] = 1.1879e+6 \quad (51)$$

B. Second Case

In our second example, we assume the argument-from-design establishes an upper bound θ on the time needed to identify the faulty component. We use this θ instead of the Δ for computing the probability of nearly coincident faults. For this example, we assume $\theta = \Delta = 5/720000$.

For this case, the trials with all transients and the trials with the permanent fault the last occurring fault join the benign class.

Table 3: Classification for second argument from design

Classification and probability For no argument-from-design	Reclassification for Argument-from-design That includes recovery time
P{ 0 faults}= 9.9824e-1	BF
P{IF1} = 1.7569e-3	BF
P{p-p and IF2} = 1.2778e-8	IF
P{p-t and IF2} = 1.2778e-7	IF
P{t-p and IF2} = 1.2778e-7	BF
P{t-t and IF2} = 1.2778e-6	BF
P{p-t-t and IF3} = 6.6556e-11	IF
P{t-p-t and IF3} = 6.6556e-11	IF
P{t-t-p and IF3} = 6.6556e-11	BF
P{t-t-t and IF3} = 6.6556e-10	BF
P{CF} = 4.3036e-11	CF

For the experiment where the argument from design includes the recovery time

$$PA\{BF\} = P\{0 \text{ faults}\} + P\{IF1\} + P\{t\text{-}p \text{ and } IF1\} + P\{t\text{-}t \text{ and } IF1\} + P\{t\text{-}t\text{-}p \text{ and } IF3\} + P\{t\text{-}t\text{-}t \text{ and } IF3\} = 9.99958e\text{-}1 \quad (52)$$

$$PA\{IF\} = P\{p\text{-}p \text{ and } IF2\} + P\{p\text{-}t \text{ and } IF2\} + P\{p\text{-}t\text{-}t \text{ and } IF3\} + P\{t\text{-}p\text{-}t \text{ and } IF3\} = 1.40691e\text{-}7 \quad (53)$$

$$PA\{CF\} = P\{CF\} = 4.30360e\text{-}11 \quad (54)$$

The upper bound for the probability of failure for the injected faults is

$$PA\{Fail \mid IF\} = [\, PA\{Fail\} - PA\{CF\} \,] / PA\{IF\} = 6.80180e\text{-}3 \quad (55)$$

The confidence level for this probability is

$$P\{misled \mid IF\} = P\{misled\} / PA\{IF\} = 7.10777e\text{-}3 \quad (56)$$

The number of trials is

$$n = [\, \log(\, P\{misled \mid IF\} \,) / [\, \log(\, 1 - PA\{Fail \mid IF\} \,) \,] = 725 \quad (57)$$

As expected, incorporating a structural element (argument-from-deign) reduces the experimental effort.

X. EXTENSIONS

When considering more realistic applications of the results above, there are synchronous and asynchronous systems. Synchronous systems operate in near lockstep in order for all the good

processors to arrive at the same result. This facilitates error detection and determining the signals sent to the actuators, but it creates an operational overhead. There is interactive consistency where information arrives from different, noisy sensors at different times, and the computers must decide on a common value to produce identical results. There is clock synchronization.

Asynchronous systems avoid the overhead, but fault detection and determining the proper control commands are more difficult. An asynchronous system might be suitable for monitoring where the goal is deciding when certain parameters are out of bounds.

We can include results from the literature such as various characterizations of faults and efficient fault-injection tools.

We can attempt extending this approach to sensors and actuators.

We can place commercial-off-the-shelf components on test. We can have a desktop or laptop monitor several components where the monitoring depends on what is observed. Determining the hard-failure rate requires only occasional checking. Determining transient occurrence and perturbations requires nearly continuous monitoring.

We can design systems using commercial components. This project would assume a failure rate for the components and determine what designs could achieve certain system reliabilities. The results of this endeavor could be used to determine what failure rate needs to be established by the test above. Successful designs would justify the experiments needed to estimate the failure rates of components.

We can examine the class of systems with maintenance-on-demand. A space station or installations on a moon or planet might use such a routine. The mathematical results in section six remain the same, but the application of the theorems and the simulation will change. If the requirement is that a plant have a small probability of failure during a ten-year period, a trial consists of simulating a ten-year period.

We can apply the global fault model to other problems. Events in the model include permanent, transient, and intermittent faults. It includes independent faults and faults correlated in space or time or both. One application includes communication systems. A fault tree approach applied at the end of an operating period only considers permanent faults, but a communication system can fail during the operating period due to transients. Communication systems are also prone to correlated faults. We can do a quantitative analysis of protocols that are designed to restore connectivity after a fault occurrence.

For any system, we can use the global fault model to consider hard deadlines for performance or reliability. The current approach for performance is to use a Markov model and compute the mean downtime, but downtimes of 1/6 of a second every minute and 1 day every year are nearly the same, while the effects can be different.

XI. SUMMARY

This paper outlines a fault-injection experiment based on simulating natural-life testing for a class of systems. The class consists of those systems obtaining reliability by redundancy with an operating period without maintenance followed by a maintenance check that removes faulty components. A trial emulates an operating period, and the paper presents a formula for how many trials are needed to establish a given reliability at a given confidence level. This approach achieves efficiency by using structural arguments that the system operates correctly if fault free, which implies we need only observe the system from fault injection to fault recovery. We inject faults according to a global fault model based on the failure rate of the components. Theorems about the global fault model give the probability distributions for how many, when, where, and what kind of faults to inject during a trial. An example reconfigurable fourplex illustrates how to apply the theorems. The example includes a fault detection routine. A program simulates choosing, injecting, and detecting the faults. The example fourplex successfully completed the required number of trials.

REFERENCES

[1] K. Astrom and B. Wittenmark, *Computer-Controlled Systems*, Prentice Hall, 1997.

[2] National Research Council 2014. *Autonomy Research for Civil Aviation: Toward a New Era of Flight*, Washington, DC: The National Academies Press.

[3] Douglas M. Chapin, *et al. Digital Instrumentation and Control Systems in Nuclear Power Plants*, National Academy Press, Washington, D.C (1997)

[4] J. Goldberg, "Challenges and Directions in Fault-Tolerant Computing," in *Computer Systems for Process Control*, Springer, 1986,

[5] Xingyu Zhaoa, Kizito Salakob, Lorenzo Striginib, Valentin Robua, David Flynna, "Assessing Safety-Critical Systems from Operational Testing," *Information and Software Technology, August 2020*.

[6] J. Carreira, D. Costa, J. Silva, "Fault injection spot-checks computer system dependability," *IEEE Spectrum*, August 1999.

[7] J. Clark, D. Pradhan, "Fault injection: a method for validating computer-system dependability," *Computer*, Volume 28 Issue 6 (1995)

[8] J. Ariat, J. Boue, Y. Crouzet, "Validation-based development of dependable systems," *IEEE Micro*, Volume 19 Issue 4 (1999).

[9] C. R. Elks, N. J. George, M. A. Reynolds, M. Miklo, C. Berger, S. Bingham, M. Sekhar, B. W. Johnson, 'Development of a Fault Injection-Based Dependability Assessment Methodology for Digital I&C Systems," NUREG/CR-7151, December 2012

[10 J. Samanta, J. Bhaumik, and S. Barman, "Compact CA-Based Single Byte Error Correcting Code," IEEE Transactions on Computers, February 2018.

[11] G. Schley, A. Dalirsann, M. Rggenbergr, N. Hatami, H Wunderlich, and M. Radetzki, "Multi-Layer Diagnosis for Fault-Tolerant Networks-on-Chip," IEEE Transactions on Computers, May 2017.

[12] V. Dumitriu, L. Kirischian, and V. Kirischian, "Run-Time Recovery Mechanism for Transient and Permanent Hardware Faults Based on Distributed, Self-Organized Dynamic Partially Reconfigurable Systems," IEEE Transactions on Computers, September 2016.

[13] Z. Alkhalifa, V. Nair, N. Krishnamurthy, J. Abraham, "Design and evaluation of system-level checks for on-line control flow error detection*," IEEE Transactions on Parallel and Distributed Systems*, June 1999

[14] G. Silberman and I. Spillinger, "Using functional fault simulation and the difference fault model to estimate implementation fault coverage," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, December 1990.

[15] G. Choi, R. Iyer, and V. Carreno, "Simulated fault injection: a methodology to evaluate fault tolerant microprocessor architectures," *IEEE Transactions on Reliability*, October 1990.

[16] C. Walter, "Evaluation and design of an ultra-reliable distributed architecture for fault tolerance," *IEEE transactions on Reliability*, October1990.

[17] G. Gil-Tomas, J. Garcia-Moran, J. Baraza-Calvo, L. Saiz-Adaid, and P. Gil-Vocente, "Injecting intermittent faults for the dependability assessment of a fault-tolerant microprocessor system," IEEE Transactions on Reliability, June 2016

[18] B. Liu and L. Cai, "Reliability evaluation for single event transients on digital circuits," IEEE Transaction on Reliability, September, 2012.

[19] F. Mireshghallah, M. Bakhshalipour, M, Sadrosadati, and H. Sarbazi-Azad, "Energy-Efficient Permanent Fault Tolerance in Hard Real-Time Systems," IEEE Transactions on Computers. October 2019.

[20] C. Stroud, "Reliability of majority voting based VLSI fault-tolerant circuits," *IEEE Transactions on Very Large Scale Integration*, December1994.

[21] C. Constantinescu, "Teraflops supercomputer: architecture and validation of the fault tolerant mechanisms," *IEEE Transactions on Computers*, Volume 49 Issue 9 (2000).

[22] F. Farahmandi and P. Mishra, "Automated Test Generation for Debugging Multiple Bugs in Arithmetic Circuits," IEEE Transactions on Computers, February 2019.

[23] J. Wang, M. Ebrahimi, L. Huang, X. Xie, G. Li, and A. Jantsch, "Efficient Design-for-Test Approach for Networks on a Chip," IEEE Transactions on computers, February 2019.

[24] F. Zadegan, D. Niklov, and E. Larson, "On-Chip Fault Monitoring Using Reconfigurable IEEE 1687 Networks," IEEE Transactions on Computers, February 2018

[25] S.-A. Hwang, J.-H. Hong, C.-W Wu, "Sequential circuit fault simulation using logic emulation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, August 1998.

[26] Hyung Ki Lee, Dong Sam Ha "HOPE: an efficient parallel fault simulator for synchronous sequential circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, September 1996.

[27] Kwang-Ting Cheng, "Transition fault testing for sequential circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, December 1993).

[28] D. Smith, B. Johnson, and J. Profeta, "System dependability valuation via a fault list generation algorithm," *IEEE Transactions on Computers*, August 1996.

[29] T. Tsai, M. Hsueh, H. Zhao, Z. Kaibarczyk, and R. Iyer, "Stress-based and path-based fault injection," *IEEE Transactions on Computers*, November 1999.

[30] C. Yount and D. Siewiorek, "A methodology for the rapid injection of transient hardware errors," *IEEE Transactions on Computers*, August 1996.

[31] J. Barton, E. Czek, Z. Segall, and D. Siewiorek, "Fault injection experiments using FIAT," *IEEE Transactions on Computers*, April 1990.

[32] W. Kao, r. Iyer, and D. Tang, "FINE: A fault injection and monitoring environment for tracing the UNIX system behavior under faults", *IEEE Transactions on Software Engineering*, Volume 19 Issue 11 (199).

[33] A. White, "Reliability with imperfect diagnostics," *Micro Electronics Reliability*, September 1984.

[34] U.S. Department of Defense, (1991) *Military Handbook, "Reliability Prediction of Electronic Equipment, MIL-HDBK-217F, 2*

[35] M. White and J. Bernstein, "Physics-of-Failure Based Modeling and Lifetime Evaluation," http://nepp.nasa.gov.

[36] M. White, "Commercial Off-The-Shelf (COTS) Parts Risk & Reliability User & Application Guide," JPL Publication 17-5.

[37] DOT/FAA, *Review of Pending Guidance and Industry Findings on Commercial Off-The-Shelf (COTS) Electronics in Airborne Systems*, August 2001.

[38] N. Leveson, "The Use of Safety Cases in Certification and Regulation," https;//dspace.mit.edu.

[39] S. Wilks, *Mathematical Statistics*, Wiley, New York, 1963.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
06/01/2021	TECHNICAL MEMORANDUM	

4. TITLE AND SUBTITLE

Establishing Fault Tolerance for a Class of Systems by Experiment

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

White, Allan L.

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

340428.01.10.07.01

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

NASA Langley Research Center
Hampton, VA 23681-2199

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

National Aeronautics and Space Administration
Washington, DC 20546-001

10. SPONSOR/MONITOR'S ACRONYM(S)

NASA

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

NASA/TM-20210013688

12. DISTRIBUTION/AVAILABILITY STATEMENT

Unclassified - Unlimited
Subject Category -Mathematical and Computer Sciences (General)
Availability: NASA STI Program (757) 864-9658

13. SUPPLEMENTARY NOTES

14. ABSTRACT

A long-standing problem in system verification is establishing fault tolerance at the ultra-high level by experiment. It is considered impossible because of system complexity and the enormous number of trials needed. This paper considers the problem for a class of digital systems that use redundancy to achieve reliability. The class is the systems that operate for a period of time without maintenance followed by a maintenance check that replaces components identified as faulty. The paper considers simulating a natural life test where a natural life test observes a number of operating periods. If the system does not fail during the test, it can be said to have a certain reliability at a certain confidence level. The approach in this paper is to make the simulated life test more efficient while maintaining realism by integrating structural arguments, information on fault occurrence, and fault injection in the lab. The major result of this paper is constructing a global fault model using the failure rate of the components and proving theorems about the model that tell how many, what kind, when, and where to inject faults. A simple example illustrates applying the theorems.

15. SUBJECT TERMS

design of experiments; fault tolerance

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			HQ - STI-infodesk@mail.nasa.gov
U	U	U	UU	34	19b. TELEPHONE NUMBER *(Include area code)* 757-864-9658