Latent Fault Markov Model for a Highly Reliable Triplex Computer System

Frederic L. Swern*
Stevens Institute of Technology, Hoboken, New Jersey
Salvatore J. Bavuso†
NASA Langley Research Center, Hampton, Virginia
and Anna L. Martensen‡
PRC Kentron, Inc., Hampton, Virginia

Current airborne digital systems are being designed around multiple microprocessors to achieve reliability through redundancy. In such a system, monitoring and voting take place at the processor output only, rather than at many points within the processor, as might have been done in the past. Under these circumstances, it is possible for latent faults to develop within each processor, degrading the overall reliability of the system. A Markov model of a triplex system was constructed taking into account this fault latency, and a quantitative assessment of its effect on system reliability is presented.

Introduction

THE advent of redundant and highly reliabile airborne digital systems has raised a number of issues in connnection with the ability of such systems to detect, isolate, and recover from hardware faults. Detection must be done in a timely manner to prevent latent faults from benignly accumulating in each channel. When faults accumulate in this manner, it is possible for a particular aircraft maneuver to activate multiple faults, resulting in system failure.

Most airborne systems employ comparison monitoring, self-testing, or a combination of both techniques to achieve the requisite detection and isolation capability. The channels are interconnected in such a manner that is is possible to detect and isolate a failure in any single channel and reconfigure the system to continue operating without that channel. In the past, the processor itself could be partitioned into smaller modules, each module being optimum for the immediate detection of faults. With the advent of microprocessors into the avionics world, the partitioning of the processor into modules for monitoring purposes is infeasible, and an architecture using the processor as a building block is used. Such an architecture is desirable from an economic standpoint and has been studied extensively in the past.[1-3]

It is apparent that the reliability of such an architecture is heavily dependent on the ability of a monitoring mechanism to handle any fault that might occur. The probability that a fault is detected by the monitoring mechanism is referred to as coverage. When coverage is low, system reliability is low, even if the number of redundant modules is large. This problem was explored by Bavuso.[4] As an example, consider a channel whose mean time between failures is 2000 h. Under an assumption of perfect coverage, it is possible to build a triplex system with a probability of system failure of 1.25×10^{-10} for a 1-h flight. However, if the coverage of the comparator in this system is only 0.99, the probability of not handling properly one faulted processor in the 1-h flight is 5×10^{-6}. Consider that the Federal Aviation Administration recommends the probability of system failure of a flight control system be on the order of 10^{-9} for an hour of commercial aircraft flight, while military standards use a failure probability of 10^{-7}/h.

If one can ensure that there is a path from every fault that might occur to the monitoring mechanism, the coverage still may not be perfect. Imperfect coverage is a result of fault propagation time from the point of occurrence to the detection mechanism. When this time is significant, the fault becomes latent. Because of the complexity of a processor and its complement of memory devices, it becomes increasingly difficult to show when, or if, any fault within the processor will reach a monitor point and be detected. Under these conditions, it is possible for faults to accumulate within the processors in such a manner that, when a particular fault finally propagates to the monitor, faults have occurred in more than one redundant channel. During the latency period of the first fault, a second fault may have occurred in another channel.

Consider, for instance, a failure in the processor memory addressing logic such that a certain block of memory is not accessible by the processor, while the remainder of memory operates in the normal manner. The probability of detection for this fault is a function of the probability of accessing that particular block of memory during the flight scenario. If this block of memory were accessed only during the landing modes, then the fault conceivably could remain latent for the full flight period of, say, 1 h. If 4% of the faults in a particular channel with a 2000-h MTBF displayed this type of latency, then the probability of system failure (due to latency) would be on the order of 2.4×10^{-9}, already double that recommended by the FAA for the system as a whole. Similarly, using special features of the processor (such as seldom-used instructions, registers, or special I/O) may also produce significant latencies. It is the responsibility of the designer to show that the effects of such latent faults truly are an acceptable factor in overall system reliability.

A variety of techniques are available to increase an unacceptably low system reliability caused by latency. All faults of

Received July 8, 1987; presented as Paper 87-2605 at the AIAA Guidance, Navigation, and Control Conference, Monterey, CA, Aug. 17-19, 1987; revision received Jan. 11, 1987.

*Associate Professor, Department of Mechanical Engineering, Member AIAA.
†Senior Research Engineer, System Validation Methods Branch.
‡Reliability Engineer.

interest must be detectable with some portion of programming that propagates the fault to a monitor point. There may exist a group of faults that have a high probability of not being propagated to a monitor during the entire mission, and may remain latent for many missions. These faults can be uncovered with a preflight self-test that ensures their absence at the start of the mission. Such testing effectively bounds the latency time of these faults (to, at most, the mission time). When latency time is still excessive, a background self-test program can be run, further bounding latency time. Designers have added a self-test in many instances to exercise seldom-used portions of hardware, improving system reliability.

A methodology is needed to determine the extent and effect of such architectural changes required to meet the desired system reliability. Some requirements for this methology are to ensure that, for all faults under consideration, there is a path that leads to a monitoring point, and the time between occurrence and detection at a monitoring point is sufficiently small. To accomplish this, the methology must include a means to measure fault detection characteristics of the monitoring system including latency time, and use these characteristics to predict system reliability. Measurement of fault detection characteristics was addressed in Ref. 8. The objective of this paper is to prevent a Markov model that can be used for the latter task.

Previous work in reliability estimation for avionics systems has often used a worst-case approach. It is assumed that a latent fault, together with any other fault, will result in system failure (sometimes referred to as "malicious excitation"). If the overall reliability under this assumption is still acceptable, it follows that the actual system should be more reliable. Bavuso[4] constructed a triple module redundant (TMR) model based on this premise. He showed that the overall reliability of a TMR system is strongly dependent on the "coverage" of both the comparator and the secondary detection means. However, as the comparator deviates slightly from perfect coverage, the reliability of the system decreases drastically. Nagel[5] and McGough and Swern[6] showed that, if one considers coverage to include only those faults detected immediately after they occur, the coverage of a comparator in triplex systems is probably considerably less than perfect. McGough[7,9] discusses the probability of a second fault occurring before recovery from the first fault is complete. Because the recovery time is small, he considers the faults to be concurrent. With latent faults, the computations are similar, except that latency time can be much longer than recovery time.

Definitions and Mathematical Background

In order to examine the effects of fault latency on system reliability, it is necessary to define a mathematical model of fault propagation for which latency parameters then may be based. The model employed was described fully in Ref. 8; however, the salient features are reproduced below to provide a background for the reader.

1) *Fault*: Let the processor be represented by a set of components $G_1,G_2,...,G_k$ and an interconnection mapping of the components in G. Each member in G has a functional model defined in terms of its inputs and outputs. A fault is defined as a malfunction of one of the members of G such that its functional model has been altered.

Each member of the set G has an associated set containing all possible alterations under consideration in a particular reliability assessment. This set is referred to as the *fault model.*

2) *Failure*: The propagation of a fault so that the ouput of the processor is erroneous will be termed a failure of that processor, a failure of the channel containing that processor, or, simply, a failure.

3) *System failure*: The propagation of an erroneous value past the comparator to a surface actuator for at least one iteration of the control program because of a failure in one or more channels of the system will be considered a system failure.

4) *Fault latency*: The time from the occurrence of a fault, as might be measured by a suitably placed test probe within the processor (if placement of such a probe were possible), until its subsequent appearance at a comparator will be called the fault latency time.

Two faults will be considered *similar* when the first erroneous value that reaches the comparator after the fault occurs is the same for both. By previous definition, the existence of two failures giving similar signals to the comparator for at least one iteration leads to a failure of the system. This is somewhat conservative because, unless the inputs to the comparator are equivalent for a large number of iterations, it is possible for the dual faults to be detected before a harmful output arrives at the aircraft surface.

The internal state of an airborne control system can be represented on a suitable state space, represented here by the vector X. In a general way, the algorithms executing on each of the redundant processors can be represented mathematically as

$$X(k+1) = g[X(k),W(k)]$$
$$Y(k) = h[X(k)] \tag{1}$$

where

W = stochastic variation due to turbulence, noise, etc.
Y = processor output and comparator input
g = function that takes the processor from its present state into its next state
h = function that relates the comparator input to the state variables

It is noted that g is a nonlinear function dependent on the processor architecture, the software, the control algorithm, and the response of the aircraft.

There exists no general form for solutions to Eq. (1). However, a transition function ϕ exists that takes X from a particular set of initial conditions at a particular instant to some future point on the flight path. Let W represent a particular noise history, and let j represent a particular iteration of the control program with associated state vector $X(j)$. Then the output at the comparator after m iterations from j can be obtained from the relation

$$Y(j+m) = h\{\phi[X(j),W,m]\} \tag{2}$$

Assume that a fault occurs at iteration j. This causes an alteration of a processor's hardware, and the algorithms that describe it are also altered. The mappings g, h, and ϕ are changed by the fault so that

$$X(k+1) = g_n[X(k),W(k)]$$
$$Y(k) = h_n[X(k)] \tag{3}$$

where g_n and h_n are the mappings for a particular fault n chosen from the space of all possible faults of the processor η. For completeness, let $n = 0 \in \eta$ be the unfaulted state.

Let n_1 and n_2 be processor faults, $n_1, n_2 \in \eta$. Let a comparator be connected between two like processors at Y—one processor containing fault n_1 and one processor containing fault n_2. Then the *detection function* is given by

$$D_{n_1}(X,W,n2,m) = 0, \quad \text{if } h_{n_1}\{\phi_{n_1}[X(j),W,m]\} = h_{n_2}\{\phi_{n_2}[X(j),W,m]\}$$
$$= 1, \quad \text{if } h_{n_1}\{\phi_{n_1}[X(j),W,m]\} \neq h_{n_2}\{\phi_{n_2}[X(j),W,m]\} \tag{4}$$

If $n_1 = 0$, then $D_0(X,W,n,m)$ is the detection function for comparing the output of a good processor to the output of a processor with fault n. Further, the *latency time of a fault* is defined by

$$T(X,W,n) = \min \{m \mid D_0(X,W,m,n) = 1\} \tag{5}$$

Let χ represent the space of all possible states of the system in a normal flight, and Ω represent the space of variations of the noise W. Then the expected value of T can be computed for each fault n as

$$\tau(n) = \sum_{k\in\chi} \sum_{1\in\Omega} P(X_k)P(W_1)T(X_k,W_1,n) \tag{6}$$

The probability of two faults delivering the same erroneous ouput to the aircraft surface is given by

$$\delta(n) = \sum_{k\in\chi} \sum_{1\in\Omega} P(X_k)P(W_1) \times \frac{\sum_{i\in\eta} p(f_n)D_0(X_k,W_1,i,1)[1 - D_n(X_k,W_1,i,1)]}{\sum_{i\in\eta} p(f_n)D_0(X_k,W_1,i,1)} \tag{7}$$

where $p(f_n)$ is the failure rate of component f_n.

TMR System Architecture

A block diagram of the TMR system under consideration is shown in Fig. 1. It contains three similar computational channels replicated from the analog sensors to the actuators that command the aircraft surfaces. Each channel is a complete flight controller that operates independently of the other two. For fault detection, a voting monitor is implemented at the output to the actuators either in hardware (analog or digital) or software. While each implementation does not change the structure of the model, it has a significant effect on the value of the model parameters. The voter is a midvalue select device that masks the fault from reaching the surface and causing erroneous control of the aircraft (which would constitute a system failure). Logic exists in either hardware or software so that, when one channel fails, it is disengaged.

After a single failure, the remaining two channels continue controlling the aircraft, comparing their outputs one against the other. When a discrepancy occurs, both processors will execute a self-test program to determine which channel has failed. If one channel passes self-test and one channel fails, then the good channel will continue controlling the aircraft and the bad channel will be disengaged. If both channels pass (or fail), the self-test, it is impossible to say which channel has failed. It is assumed that one channel is good and one channel is bad; the inability of the self-test to discern between them is because of its imperfect coverage. It is equally probable that either channel is good. The model reflects this mechanism: when self-test coverage fails, one channel will be chosen arbitrarily to control the aircraft and the other will be disengaged.

A Markov model of this system with latent faults included was constructed for the purposes of estimating the probability of system failure. However, new latency parameters were required to construct the transition rates between some of the Markov states. State transition in the Markov model occurred as a result of the execution of a single iteration of the control algorithm. Let

$$\eta_t(X_k,W_k) = \{n \mid n\in\eta,\ T(X_k,W_k,n) = t\} \tag{8}$$

represent the set of faults that are detected at a particular iteration t, given that the system started in state X_k and experienced noise W_k. The probability of detection of a fault at t under the same conditions is given by

$$\gamma_t(X_k,W_k) = (1/\lambda) \sum_{n\in\eta_t} p(f_n) \tag{9}$$

where λ is the failure rate of a single processor. The probability that a fault is detected (given that it has occurred) within a certain number of iterations, α, is

$$C1(X_k,W_k,\alpha) = \sum_{t<\alpha} \gamma_t \tag{10}$$

The Markov model requires the rate of propagation of a fault to a monitoring point per control program iteration to compute the transition probability from a state with a latent fault to the ones in which fault recovery has been attempted. This rate, denoted by Γ, is a function of t, which represents the time after which the fault has occurred. A more complex Markov diagram could be constructed to account for this dependence on t. For the purpose of studying the effect of Γ on system failure rate with a similar model, it was estimated by

$$\Gamma(X_k,W_k) \le \max_{t>\alpha} \{\gamma_t/t\} \tag{11}$$

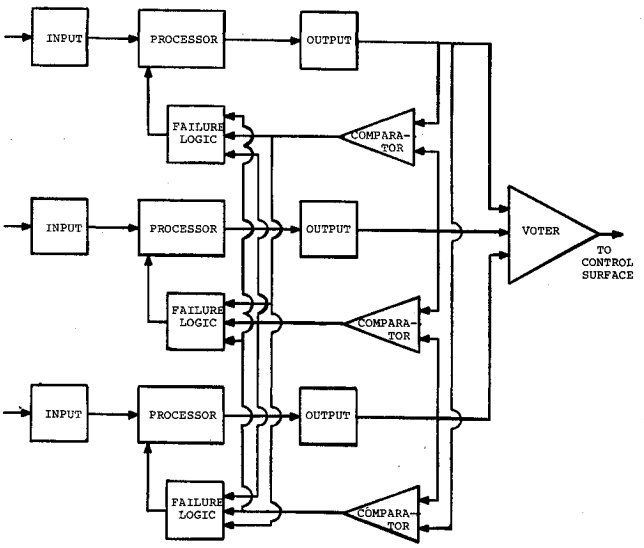

Fig. 1 TMR architecture.

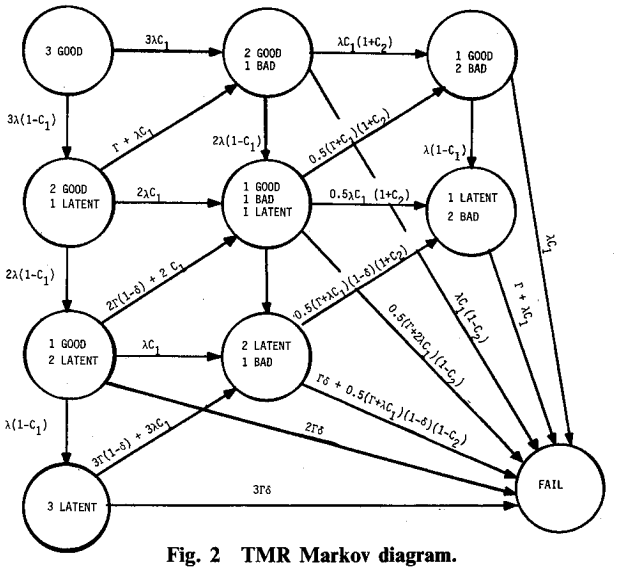

Fig. 2 TMR Markov diagram.

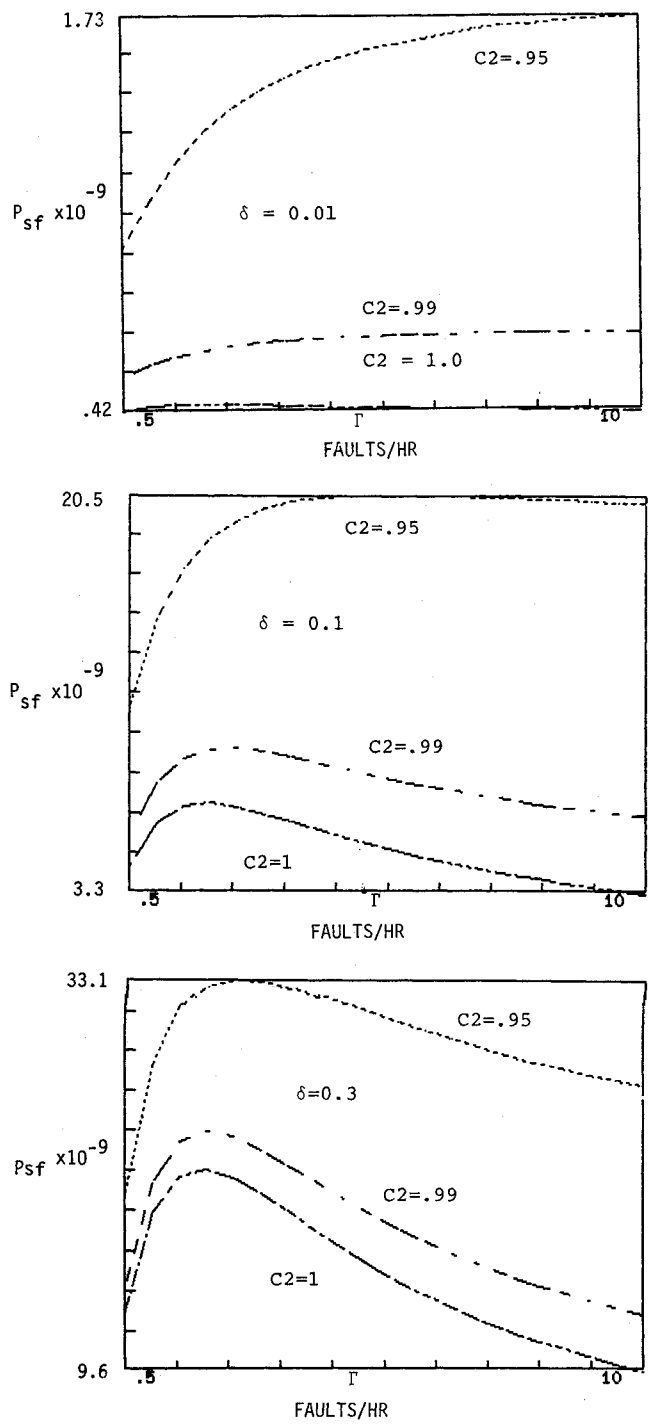

Fig. 3 TMR study results.

If the aircraft is flying in a cruise condition, then X is not expected to change significantly from iteration to iteration of the processor. Then both $C1$ and Γ should be relatively constant over the flight. If the noise W is expected to be purely random, then it too will cause relatively small variations in $C1$ and Γ.

The remaining factor that might adversely affect the probability of system failure in a significant manner is a change in flight envelope X that one might obtain when the aircraft enters, for example, a landing mode. The implication here is that failures that are not detected during the cruise portion of the flight might accumulate, although they would remain benign. However, when switching to autoland, the values of $C1$ and Γ might change in such a manner that the overall probability of failure of the system is increased significantly.

Construction of the Markov model is completed by selecting for each set of values of X and W (i.e., each flight regime) corresponding values for $C1$ and Γ. These parameters, along with the other parameters defined previously, are used to form the transition probabilities in the model. Since X and W are functions of time with respect to the beginning of the flight, the model is still Markov; however, it is nonhomogeneous. Figure 2 shows the model used for analysis in the studies performed.

Markov Model Evaluation

In order to understand more fully the implications of latent failures, the Markov model was evaluated using parameters that represented a typical avionics processor. The model given in Fig. 2 has transition probabilities that are functions of five parameters:

λ = single channel failure rate
$C1$ = comparator primary software coverage as defined in Eq. (10)
$C2$ = secondary detection means coverage
Γ = propagation rate of latent failures
δ = probability that two latent failures are excited and produce the same erroneous output at the comparator

While many of the transition probabilities in the model are straightforward, the transition from states with one bad processor to those with two bad processors requires some comment. Recall that, when self-testing cannot identify the failed processor, one of the two processors is chosen arbitrarily to control the aircraft. In terms of the preceding parameters, the probability of transitioning from, say, a state with two good processors with one bad processor to one with two bad processors and one good processor is

$$2\lambda\, C1\, C2 \text{ (for faults picked up by self-test)}$$
$$+2\lambda\, C1\,(1-C2)0.5 \text{ (for faults not picked up by self-test)}$$
$$=\lambda\, C1\,(1+C2)$$

A value for λ of 0.0005 failures/h was used to evaluate the transition parameters, as this was considered typical of avionics processors. From previous studies,[5,6,8] a value for $C1$ of 0.5 was considered reasonable with $\alpha = 1$.

Values for the other parameters were not as easy to estimate without a specified processor and software in mind. Therefore, $C2$ and δ were varied, taking on three discrete values: $C2$: 0.95, 0.99, 1.0; δ: 0.3, 0.1, 0.01. The parameter $C2$ represents the coverage of a secondary detection means, which often is a self-test program. The coverage goal of such programs usually is 95% or higher; hence, the three values were chosen for $C2$. The value of δ was unknown and three values were chosen as shown earlier. However, studies showed[8] that a reasonable value for δ was around 0.07, which concurs with the region of δ chosen. Because the value of Γ was completely unknown, it was allowed to vary over its entire range.

The resulting probability of system failure was plotted as a function of Γ in Fig. 3 for the various parameter combinations.

The results can be interpreted from architectural considerations. A propagation rate of latent faults of zero implies that $(1.0 - C1)$ of the possible hardware faults never affect the operation of the processor. Effectively, the processor operates using less hardware, and the resultant probability of failure is smaller. On the other hand, if the propagation rate of latent faults is very high, it is as if no fault ever becomes latent and as if $C1 = 1.0$. Thus, the two end points of the curves are easily predictable, neglecting the dynamics of latency.

In the middle region of Γ, its effect depends on whether or not two latent faults are likely to have the same erroneous output. When this is not the case, latency actually improves the survivability of the system because a certain percentage of the faults may remain latent until the end of the flight. When the probability of two latent faults giving the same erroneous output is high, system performance suffers dramatically. However, there is a latent failure propagation rate that is most disastrous for the system, and this rate can be calculated based on the other parameters.

Conclusions

Evaluating the probability of system failure using a Markov model as a function of the propagation of latent faults has brought to light a number of characteristics of fault latency that might be of use in improving system designs. In summary, the following conclusions were drawn:

1) If the propagation rate of latent faults is extremely high, they do not significantly affect the probability of system failure.

2) If the propagation rate of latent faults is extremely low, the survivability of the system is improved.

3) There is a particular propagation rate that is most harmful to the survivability of the system, and this rate is a function of the duration of the flight.

4) If the probability of any two faults giving the same output is extremely low, then the probability of system failure caused by latency is decreased significantly.

Acknowledgments

The first author wishes to acknowledge NASA Langley Research Center for support of this work under Grant NAG-1-587. In addition, thanks go to Michael Masiello, who helped with much of the computer work. Contributions by John McGough are also greatly appreciated.

References

[1]Hopkins, A. L., Smith, T. B., and Lala, J. H., "FTMP—A Highly Reliable Fault-Tolerant Multiprocessors for Aircraft," *Proceedings of the IEEE*, Vol. 66, Oct. 1978, pp. 1221–1239.

[2]Ramamoorthy, C. V. and Han, Y., "Reliability Analysis of Systems with Concurrent Error Detection," *IEEE Transactions on Computers*, Vol. C24, Sept. 1975, pp. 868–878.

[3]Hopkins, A. L. and Smith, T. B., "The Architectural Elements of a Symmetric Fault-Tolerant Multiprocessor," *IEEE Transactions on Computers*, Vol. C24, May 1975, pp. 498–505.

[4]Bavuso, S. J., "Impact of Coverage on the Reliability of a Fault Tolerant Computer," NASA TN D-7938, Sept. 1965.

[5]Nagel, P., "Modeling of a Latent Fault Detector in a Digital System," NASA CR 145371, Aug. 1978.

[6]McGough, J. G. and Swern, F. L., "Measurement of Fault Latency in a Digital Avionic Miniprocessor," NASA CR 3462, Oct. 1981.

[7]McGough, J. G., "Effects of Near-Coincident Faults in Multiprocessor Systems," *Proceedings of the AIAA/IEEE Digital Avionics System Conference*, AIAA, New York, 1983, pp. 16.6.1–16.6.7.

[8]Swern, F. L., Bavuso, S. J., Martensen, A. L., and Miner, P. S., "The Effect of Latent Faults on Highly Reliable Computer Systems," *IEEE Transactions on Computers*, Inst. of Electrical and Electronic Engineers, New York, Aug. 1987.

[9]McGough, J. G., Smotherman, M., and Trivedi, K. S., "The Conservativeness of Reliability Estimates Based on Instantaneous Coverage," *IEEE Transactions on Computers*, Vol. C34, July 1985, pp. 602–609.