Safety-Reliability of Distributed Embedded System Fault Tolerant Units

Juan R. Pimentel
Kettering University
1700 W. Third Ave.
Flint, Michigan USA
jpimente@kettering.edu

Abstract *– In this paper we compare the relative performance of two fault tolerant mechanisms dealing with repairable and non-repairable components that have failed. The relative improvement in the reliability and safety of a system with repairable components is calculated with respect to the corresponding system where the components are not repairable. The fault tolerant systems under study correspond to a flexible arrangement of fault tolerant units (FTU's) suitable for dependable distributed embedded systems. A simple simulation-based methodology to numerically evaluate dependability functions of a wide variety of fault tolerant units is used. The method is based on simulation of stochastic Petri Nets. A set of 15 FTU configurations belonging to five groups is analysed. The methodology allows a quick and accurate evaluation of dependability functions of any distributed embedded system design in terms of the type of FTU (i.e., node or application), replicas per group, replicas per FTU, with or without repair functionality, and shared replicas.*

Keywords*: Distributed systems, fault tolerance, dependability, real-time systems, reliability, safety, simulation, stochastic Petri-Nets.*

I. INTRODUCTION

In a previous paper [12], we have presented results of reliability evaluation of a system composed of a flexible arrangement of fault tolerant units when replicas fail without the possibility of being repaired. In this paper, we extend the results of [12] to include the situation of a replica being repaired as soon as it fails.

There are several ways to perform the repair process. If the failure is due to a physical component failure then the component can be replaced. If the failure is due to a design error, then appropriate actions can be effected. For example, a new version of the control software can be programmed into the ECU's. The latter is particularly appropriate to correct the so-called control failures. Examples of control failures include [13]:

- A required event that does not occur.
- An undesirable event
- An incorrect sequence of desired events
- Two incompatible events occurring simultaneously
- Timing failures in event sequences
- Exceeding maximum time constraints between events
- Failing to ensure minimum time constraints between events

We make the following general assumptions regarding the system under study: First, to deal with control failures listed above an improved version of the software exists, second, we assume no specific way of performing the repair. The repair could happen on-line (automatic), via the Internet, or at a service shop. Thus examples of repair actions include changing a new microcontroller chip or installing an improved version of a program.

The motivation for studying systems with repairable components is that they are important for the so-called safety critical systems. It is well known that a system's reliability and safety can be improved if redundant components are used. Furthermore, if some components are repairable after they fail, both the system's reliability and safety can be improved even further. A second motivating factor is that because of continued advances in memory and programming (e.g., bdm, large flash memories) , it is relatively easy to reprogram a microcontroller and deal with control failures in this way. A final motivating factor is our desire to develop a relatively simple tool to evaluate dependability and safety functions of various fault tolerant designs of distributed embedded systems in a relatively easy fashion.

II. SAFETY CRITICAL SYSTEMS

To illustrate some safety and reliability issues of fault tolerant units in distributed embedded systems we consider the following example involving a software architecture for a steer-by-wire system.

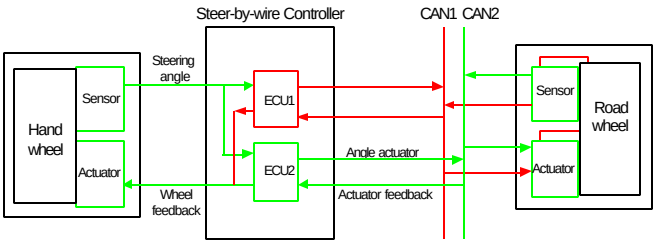

Figure 1. Hardware architecture of a steer-by-wire system.

A steer-by-wire system is a complex system involving a number of sensors, actuators, communication devices, electronic control units (ECU's), signal processing and conditioning devices, etc. The system gets more complicated if more stringent requirements (e.g., better reliability and safety) are to be taken into account. Developing software for such systems is challenging and the software design process can be simplified if it is based on an appropriate architecture.. We have purposely simplified the functionality of actual steer-by-wire systems to illustrate the architecture and associated software design issues. A primary design requirement of the architecture is that of safety and reliability. The main mechanism used to deal with safety and reliability is that of redundancy of safety-critical components. Another mechanism is the possibility of repair of some failed components.

As depicted in Fig. 1, the hardware architecture of a steer-by-wire system is made up of a hand-wheel unit, a road wheel unit, the steer-by-wire controller, and the redundant CAN communications (CAN1 and CAN2) [3, 9]. We assume that there is only one sensor (S1) and one actuator (A1) at the hand-wheel which are directly wired to both ECU's that make up the redundant steer-by-wire controller. We also assume that an ECU has only one CAN controller thus each ECU and its corresponding CAN bus controller can be treated as a single "line replaceable unit" (LRU), (i.e., a failure of the CAN bus controller is equivalent to a failure of the corresponding ECU and conversely). We further assume that there are redundant sensors and actuators at the road-wheel unit. Thus it is implied that there are two redundant microcontrollers (MC3 and MC4) at the road wheel unit handling the sensing and actuating functions. The primary assumption in the redundant configuration is that when one component of a redundant configuration fails, the other component will take over the function of the failed component.

Assuming that ECU2 is the operational (primary) controller, the system operates as follows: upon a driver command from the steering wheel, the sensor in the hand wheel will send the corresponding signal to both ECU's. The two ECU's will perform their functions asynchronously and output their messages on both CAN buses. However, the road wheel actuator (A2) will only read from the CAN2 bus to generate signals for the road-wheel motor. Likewise, the road-wheel sensor (S2) will use the CAN2 bus to send actuator feedback information to ECU2 which in turn will forward the information to the actuator in the hand-wheel.

Failure modes

From a safety standpoint one can identify a number of hazardous states (failure modes) and focus the analysis on each of these states separate from one another. This is so because the concern is on safety rather than correctness. That is, a system is *safe* it is free from *mishaps* even if it does not accomplish its mission or functional objectives [13].

Based on the drive-by-wire system and its architecture, we can distinguish the following failure modes:

1. The road-wheels do not respond to a command from the hand-wheel. (F-FORW)
2. The road-wheels turn by themselves without any command from the driver. (F-AUT)
3. There is no road feedback to the driver. (F-FDB)

Figure 2 depicts a logic diagram for states of the system for two failure modes: F-FORW, and F-FDB. Assuming that each system state x = 1 when the system (or component) is operational, and x = 0 when it has failed,

F-FORW = (CAN1 v CAN2) ^ ((ECU1 v ECU2)^(MC3vMC4)) ^ (S1 ^ A2)

One of the main advantages of distributed embedded systems is the potential to operate even under the presence of some faults through the use of redundant units configured as special fault tolerant units (FTU). This feature is of tremendous advantage for applications requiring a high level of safety and reliability (e.g., passenger cars, airplanes, etc.) [1, 2, 5]. In figure 2, CAN1 and CAN 2 constitute a fault tolerant unit. Likewise, ECU1 and ECU2 constitute another fault tolerant units. More complex fault tolerant units are discussed in the next section. It is important to note that the assumptions made to illustrate the steer-by-wire architecture are for the purpose of explanation only. More detailed assumptions of the model used and system under study are made in the sections that follow.

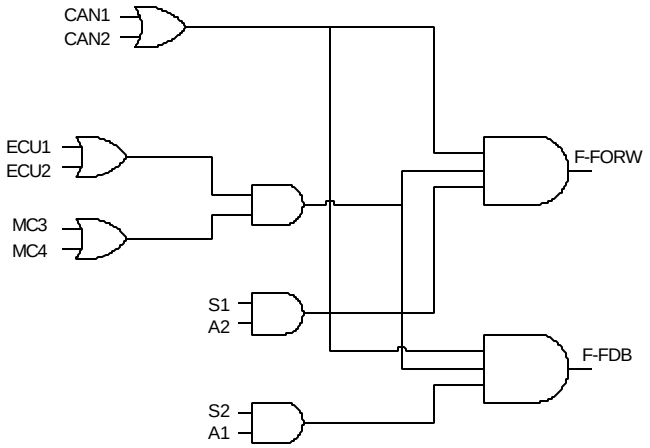

Figure 2. Logic diagrams corresponding to two failure modes of the steer-by-wire architecture.

III. RELIABILITY AND SAFETY ISSUES

The notion of dependability involves the reliance of a system on the quality of services it provides during an extended interval of time [4]. Important attributes of dependability include reliability, and safety. **Reliability** is the probability of a system staying in the operating state without failure, and **safety** is the probability of a system staying in safe states despite component failures. In the following we provide more precise definitions of these functions.

Let $S = \{s_i, i \in I\}$ be the set of all possible states of a system. We can divide S into two disjoint subsets S_s and S_f where S_s denotes a subset of states where the system is operating sucessfully and S_f where the system has failed. Thus

$S = S_s \cup S_f$

$S_s = \{ s_i, i \in I_s\}$, and $S_f = \{ s_i, i \in I_f\}$

Based on the previous definitions we provide the following definitions.

Reliability:

$R(t) = Pr\{s(t) \in S_s / s(t) \in S_s, t \in [0,t>\}$

MTTF (Mean time to failure): $MTTF = \int_0^\infty R(t)dt$

To define safety, we decompose S_f into two states **B** and **C**. The original set of sates S_s is renamed **A**.

Safety:

$S(t) = Pr\{s(t) \in (\mathbf{A},\mathbf{B})/ s(t) \in (\mathbf{A},\mathbf{B}), t \in [0,t>\}$

From the reliability and safety definitions we can see that the evaluation of the reliability or safety functions are similar, differing only in the underlying states. In [11], this approach has been used to evaluate the dependability of fieldbus networks.

A. Redundancy for safety and reliability. Most approaches to fault tolerance rely on extra elements introduced to detect and recover from faults.These elements are *redundant* in that they are not strictly required for the system to operate. Care must be taken when introducing redundant components to ensure that they do not lead to a *less* reliable system. A fault tolerant system typically go through the following phases: error detection, damage assessment and confinement, error recovery, and fault treatment. No fault tolerant scheme can start to operate until the fault has manifested itself as an error, which can subsequently be detected, thus the need for *error detection.* Once an error has been detected, the consequences of that error must be assessed through *damage assessment and confinement*; the longer the time delay between occurrence and detection, the greater the possibility of system corruption. The aim of *error recovery* is to transform the system into a state where it can continue to provide full, or degraded, functionality. Finally under *fault treatment*, errors are viewed as the symptoms of faults that unless the cause is treated, errors may be repeated.

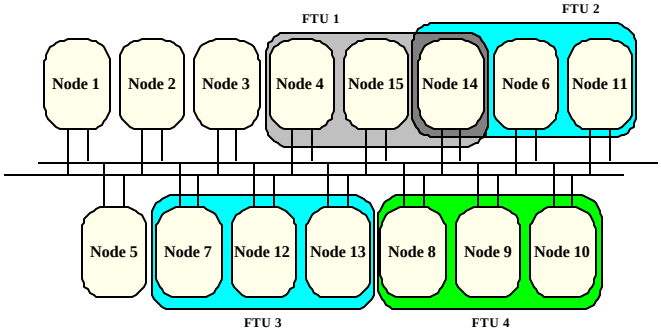

Fig. 3. Node oriented FTU's.

B. Fault Tolerant Units. As the name implies, a FTU is a unit that continues to operate even in the presence of some faults. The primary mechanism used by a fault tolerant unit is replication of software, hardware, information, and time. In this paper, we consider only dynamic redundancy (i.e., passive replication) where a normally working node is considered to be a primary node and the redundant nodes are called the secondary nodes. We use the terms redundant node, secondary

node, replica, or backup as meaning the same. In addition we are primarily concerned with hardware replication at the node level in the context of a communication network. In this context, we can distinguish two types of FTU's, *node oriented* and *application oriented*. The unit of replication in a node oriented FTU are nodes that are independent from applications. When the primary node fails only nodes that are designated replicas of the primary nodes will take over. A mechanism is needed to allow just one replicated unit to take over to avoid collisions or contention. Thus, the secondary nodes are assumed to have similar or identical functionality of the node they intend to replace. The unit of replication in an application oriented FTU are also nodes but unlike node oriented FTU's, they belong to a common pool of redundant nodes defined for a specific application. When the primary node fails only nodes that are designated replicas of the application to which the failed node belongs will take over. Figure 3 depicts four node oriented FTU's where FTU's 1 and 2 share node 14

C. FTU Configurations. Table I shows 5 groups of 15 FTU configurations with each group having 3 FTU's denoted by A, B, and C. We assume that all FTU's in the same group belong to a common application. Groups 1 and 2 contain FTU's that belong to the node oriented category whereas groups 3 through 5 contain FTU's that belong to the application oriented category. For each FTU, the node with an asterisk denotes the primary node whereas the remaining nodes are secondary (i.e., backup or replica) nodes. For example, for FTU A of group 1, node 11 is the first backup and node 12 is the second backup. The main thing to notice between the node oriented category and application oriented category is that while the former may share some replicas, the latter have a common set of replicas shared by all FTU's in the group. One can see that the total number of replicas in group 1 is 6 (two per each FTU) whereas the total number of replicas in group 2 is 3, a saving of three replicas when compared with group 1. In summary, table I shows 15 FTU's belonging to 5 groups (i.e., applications) and 3 FTU categories; the number of replicas per group vary between 1 and 6, the number of replicas per FTU vary between 1 and 3 and the number of shared replicas vary between 0 and 3. It can be noticed that there is a significant saving in the total number of replicas per group of application oriented FTU's (i.e., groups 3, 4, and 5) when compared to node oriented FTU's (i.e., groups 1 and 2).

IV. MODELS AND EXPERIMENTS

We have chosen stochastic Petri-Nets (SPN) as the mathematical framework to evaluate the reliability functions. The advantages of SPN for performance and reliability analysis of communication networks have been widely documented in the literature [6,7]. In addition, we have chosen a SPN simulator as the computational tool to evaluate the reliability functions. The main advantages of such tool is that it supports modeling a wide variety of probability distribution functions and Petri Net extensions (e.g., the test arc) and that the solution is obtained through simulation thus allowing the analytical evaluation of any model regardless of its complexity.

Table I. Distribution of nodes per groups and FTU's

Gr	FTU A	FTU B	FTU C	RPG	RPF	SR
1	5*, 11, 12	6*, 13, 14	7*, 15, 16	6	2	None
2	5*, 11, 12	6*, 11, 13	7*, 12, 13	3	2	11, 12, 13
3	5*, 11, 12, 13	6*, 11, 12, 13	7*, 11, 12, 13	3	3	11, 12, 13
4	5*, 11, 12	6*, 11, 12	7*, 11, 12	2	2	11, 12
5	5*, 11	6*, 11	7*, 11	1	1	11

(*) Denotes primary node within an FTU. Gr: Group, RPG: Replicas per group, RPF: Replicas per FTU, SR: Shared replicas.

As noted, the reliability function of an FTU is the probability that the FTU continues to operate given that it is operational. The event denoting that an FTU continues to operate is the same as the event that the primary node of the FTU or any of its backups continue to operate, according to the error detection and recovery scheme of the fault tolerant mechanism in question. We have configured a generic Petri-Net model that follows the structure of Fig. 4 where a token in the FTU-test place represents the initial operating state of an FTU. The Petri-Net model has been designed to simulate an experiment to observe the behavior of the system. Although the distribution of tokens in the model (i.e., its marking) represent the state of the primary node and all of its backups, we are only interested in two behavioral states indicating whether the FTU under test has failed or not. Depending upon the distribution of node failures, and the nature of the fault tolerant mechanism, after a certain simulated time interval T, the FTU may find itself in either of two states, a success state (the FTU continues to operate) or a failure state (the FTU is not able to provide its services). The interval T is controlled by the timing control block of Fig. 4 and the experiment is repeated N times. In each trial of the experiment, one FTU is tested (one token leaves the FTU-test place of Fig. 4) and either of two events happen; it continues to operate (one token is added to place Ns) or it fails (one token is added to place Nf). After N trials of the experiment,

the number of tokes in place Ns correspond to the number of times the FTU under study did not fail (i.e., a success) and the number of tokes in place Nf correspond to the number of times the FTU failed. A reliability value for a fixed value of T is simply the probability of finding the system in place Ns after a simulated time of T. The reliability function over time is obtained by varying T [8].

Nodes with repair

The model category correspond to a fault tolerant mechanism that involves an error detection and recovery scheme consisting of the primary node and a number of secondary nodes where only the first node that fails can be repaired. Once the primary node fails, we assume that the repair process will start immediately and it will also complete immediately. In most real systems, a repair process approximately follows a uniform distribution function with a uniform probability density function (pdf) in the interval $[r_a, r_b]$. For highly dynamic systems such as distributed embedded systems, the uniform pdf models repair processes more accurately than the traditional assumption of exponential repair density functions because the repair time is a fairly fixed value that includes a small variation. However, the values r_a, and r_b are so small as compared with the period between failures that they do not have any effect on the models.

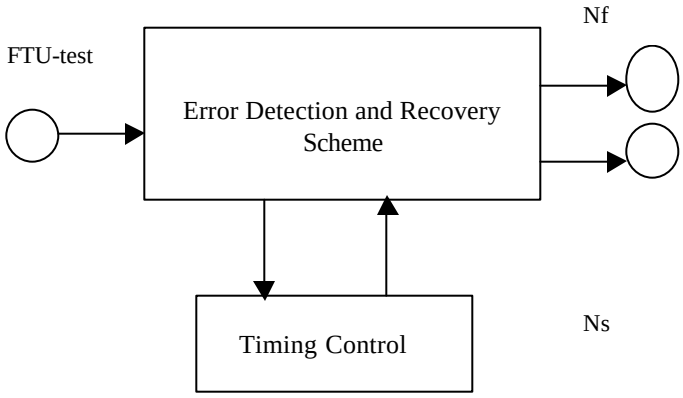

Fig. 4. Structure of a generic Petri Net model for reliability calculations

The degree of fault tolerance offered depends on the number of backups (i.e., replicas), whether the backups are shared, whether failed components can be repaired, and how backups are shared (node oriented or application oriented). As noted, the reliability function of an FTU is the probability that the FTU continues to operate given that it is operational. For the models in this category, the event denoting that an FTU continues to operate is the same as the event that the primary node of the FTU, or any of its backups, or the repaired node is operating subject to the constraint that some (if any) backup nodes may be shared with other FTU's as shown in Table I. We assume that nodes can be repaired soon after they fail but before the system fails completely.

The Petri Net model for FTU A of group 1 is depicted in Fig. 5 and corresponds to an FTU with one primary node and two backup nodes. The places on the very top of the figure (P-to-be-repaired, Prim-repaired, and P26) models the repair of the "Primary" node. The places in the intermediate section of the figure represent states indicating when the primary node is active and when it fails (Prim-failed), when the first backup is active and when it fails (bk-1-failed), and when the second backup is active and when it fails (bk-2-failed). The transitions simulating failures follow exponential distributions with mean node failure rate of 1 failure per 10^5 hours (i.e., 1 failure per 11.446 years). The transitions in black represent instantaneous transitions and the remaining transitions represent timed transitions. The number of tokens in the places Nf and Ns on the far right represent the number of failures and successes in N trials of the experiment where N is the initial number of tokens in place FTU-test. Places P15 and P16 along with transitions T21 and T22 represents the timer control block and model a timer that regulates a time window that controls the length of the experiment. The model is independent of the actual protocol used (e.g., CAN, TTP/C, or FlexRay) because of the relative high values of mean time to failures relative to the timing parameters of any specific protocol (e.g., TDMA slot time of TTP/C).

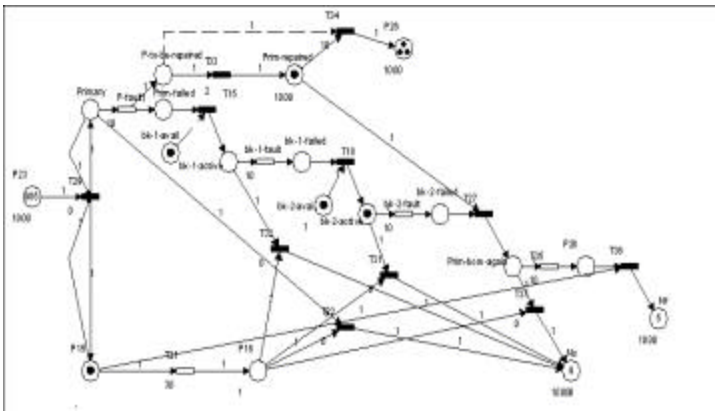

Fig. 5. Petri Net model corresponding to FTU A, group 1.

V. RESULTS

The results of the experiments are a set of reliability functions and the mean time to failure (MTTF) for all FTU configurations for models in both categories: nodes without repair and nodes with repair. In [12], the reliability functions and associated MTTF for nodes without repair were evaluated. For comparison purposes, Fig. 6 depicts the reliability functions of groups 1 and 5 for both cases, with repair and without repair. It can be noticed that the difference between MTTF values of groups with repair and without

repair (the area under the corresponding curves) is higher as the number of backups increases. For example, the MTTF of group 1 increases from 2.59x 10^5 hours (without repair) to 3.55 x 10^5 hours (with repair).

VI. SUMMARY AND CONCLUSIONS

One of the main advantages of distributed embedded systems is the potential to operate even under the presence of some faults through the use of redundant units configured as special fault tolerant units (FTU). Fault tolerant units are effective means to improve the safety and reliability of distributed embedded systems. A simple method to numerically evaluate safety-reliability functions of a wide variety of fault tolerant units has been developed. The method is based on simulation of timed Petri Nets with extensions such as the test arc. A set of 15 FTU configurations belonging to two categories has been analyzed. The degree of fault tolerance is determined by the number of backups (i.e., replicas), whether the replicas are shared, how replicas are shared in each category (node oriented or application oriented), and whether automatic repair of failed nodes is considered. The MTTF of FTU's with repairable nodes increase with respect ot FTU's having nodes without repair. Thus the reliability and safety of a system with repairable components is better than the corresponding system where the components are not repairable. The methodology presented in this paper allows the evaluation of safety-reliability functions of any distributed embedded system design to be performed quickly and accurately.

VII. ACKNOWLEDGEMENTS

The author wishes to thank the Fulbright Commission of the U.S. and Colombia for funding part of the grant under which this research was performed.

VIII. REFERENCES

[1] Bretz, E.A., "By-Wire Cars Turn the Corner," IEEE Spectrum, Vol. 38, No. 4, pp.68-73, April 2001.

[2] Poledna S., *Fault-Tolerant Real-Time Systems*, Kluwer Academic Publishers, 1996.

[3] L. Lawrenz, *CAN System Engineering: From Theory to Practical Applications*, Springer, 1997.

[4] H. Kopetz, G. Grunsteidl, "TTP – A protocol for Fault-Tolerant Real-Time Systems". IEEE Computer, pages 14-23, January 1994.

[5] Kopetz H., *Real-Time Systems, Design Principles for Distributed Embedded Applications*, Kluwer Academic Publishers, 1997.

[6] Molloy, M., "Performance Analysis Using Stochastic Petri Nets," IEEE Trans. On Computers, Vol. C 31, Sept. 1982.

[7] Narahari Y., and Viswanadham N., "Performance Modelling of a Fault-tolerant real-time Multiprocessor Using Stochastic Petri Nets", in *Reliability and Fault-Tolerant Issues in Real-Time Systems*, Proceedings in Engineering Sciences, Vol. 11, pp. 187-208, 1987.

[8] Billinton R., and Allan R. N., *Reliability Evaluation of Engineering Systems, Concepts and Techniques*, 2nd Ed., Plenum Press, New York, 1992.

[9] http://www.flexray-group.com

[10] Pimentel, J.R, and Sacristan, T. "A Fault Management Protocol for TTP/C," Proc. IECON'01 (Int. Conf. On Industrial Electronics), pp. 1800-1805, Denver, CO., Nov. 2001.

[11] Carvalho, A., and Portugal, P., "On Dependability Evaluation of Fieldbus Networks: A Permanent Fault Analysis," Proc. IECON'01 (Int. Conf. On Industrial Electronics), pp. 299-304, Denver, CO., Nov. 2001.

[12] Pimentel, J.R, and Salazar, M. "Dependability of Distributed Control System Fault Tolerant Units," Proc. IECON'02 (Int. Conf. On Industrial Electronics), Seville, Spain., Nov. 2002.

[13] Leveson, N. G., Safeware: System Safety and Computers, Addison Wesley, 1995.

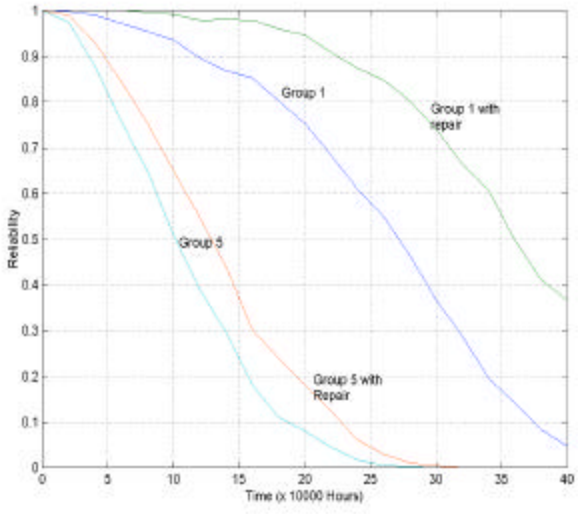

Fig. 6 Reliability functions for FTU's with and without repair.