THE CONSENSUS PROBLEM IN FAULT-TOLERANT COMPUTING

Michael Barborak[1], Miroslaw Malek[2],
and Anton Dahbura[3]

Department of Computer Sciences
University of Texas at Austin
Austin, Texas 78712-1188

TR-91-40 December 1991

---

[1] Department of Electrical and Computer Engineering, University of Texas, Austin, Texas 78712.
[2] Department of Electrical and Computer Engineering, University of Texas, Austin, Texas 78712. Currently on leave at the Office of Naval Research in London.
[3] Motorola, Inc., Cambridge Research Center, 1 Kendall Square, Bldg. 200, Cambridge, Massachusetts 02139.

The Consensus Problem in Fault-Tolerant Computing

Michael Barborak, Miroslaw Malek[1]

Department of Electrical and Computer Engineering, University of Texas, Austin, Texas 78712

Anton Dahbura

Motorola, Inc., Cambridge Research Center, 1 Kendall Square, Bldg 200, Cambridge, Massachusetts 02139

Abstract

The consensus problem is concerned with the agreement on a system status by the fault-free segment of a processor population in spite of the possible inadvertent or even malicious spread of disinformation by the faulty segment of that population. The resulting protocols are useful throughout fault-tolerant distributed systems and will impact the design of other decision systems to come. This paper surveys research on the consensus problem, compares approaches, outlines applications, and suggests directions for future work.

Categories and Subject Descriptors: C.2.3 [**Computer-Communication Networks**]: Network Operations — *network management, network monitoring*, C.2.4 [**Computer-Communication Networks**]: Distributed Systems — *distributed applications, network operating systems*, D.4.5 [**Operating Systems**]: Reliability — *fault tolerance*

General Terms: Algorithms, Design, Reliability

Additional Key Words and Phrases: consensus problem, system diagnosis, Byzantine agreement, decision theory

---

[1]Currently on leave at the Office of Naval Research in London.

Contents

INTRODUCTION

It has long been the goal of system designers to connect independent computer resources together to create a network with greater power and availability than any of its parts. Unfortunately, the reverse can happen if faulty resources are allowed to corrupt the network. In the area of fault-tolerant computing, the *consensus problem* is to form an agreement by the fault-free members of the resource population on a quantum of information, which could be a list of the faulty members, in order to maintain the performance and integrity of the system. The proposed approach is to diagnose and contain faults at the system level. Work in the area has increased with the proliferation of distributed systems that range from small, local-area networks to large, real-time, fault-tolerant systems such as that proposed by IBM to fulfill Federal Aviation Administration air-traffic control requirements [Cristian 1990]. Consensus solutions give a convenient and, sometimes, vital picture of the condition of the network.

This paper surveys over 20 years of research on this consensus problem. Section 2 examines work on *system diagnosis*, which has sprung from the seminal research done by Preparata *et al.*, and on the *Byzantine Generals Problem* introduced by Pease *et al.* [Preparata *et al.* 1967; Pease *et al.* 1980]. Sections 3 and 4 discuss how faulty processors are characterized and how they might be detected with testing. Section 5 looks at extensions to the basic work done on the consensus problem. Section 6 is a comparison of system diagnosis and Byzantine Generals Problem solutions. Section 7 is devoted to applications of consensus protocols, and Section 8 suggests directions for future studies.

2 FORMULATING THE CONSENSUS PROBLEM

The simple idea of consensus is to share information among a group of processing elements (PEs) preferably in a fault-tolerant manner. That is, the fault-free members of the PE population should be able to consistently agree on and produce correct results despite the actions, malicious or not, of the faulty segment of the population. The importance of the problem stems from its omnipresence. This problem is at the core of protocols handling synchronization, reliable communication, resource allocation, task scheduling, reconfiguration, replicated file systems, sensor reading, and other functions.

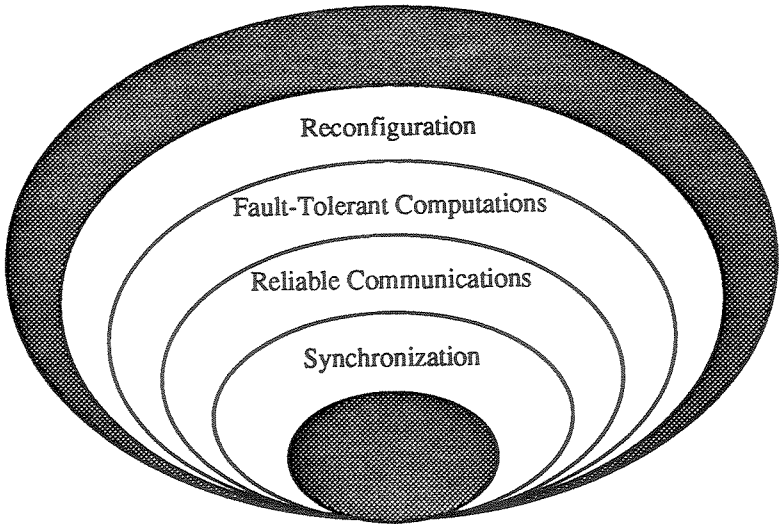


Figure 1: Consensus Problems in Fault Management.

A distributed operating system shows the abundant need for consensus procedures. Fig. 1 shows a general layered approach to fault management in which higher layers are dependent on lower layers to produce a fault-tolerant system [Malek 1991]. That is, the synchronization layer allows processors to recognize untimely messages, and to order timely messages in implementing reliable communications. Reliable communications let fault-

free processors pass fault-free messages that are used to agree on correct computations. Finally, the ability to agree on a single value allows the fault-free processors to agree on a reconfiguration after a fault. Each of these layers is a separate consensus problem.

First, the synchronization level maintains a global timepiece which is simply a consensus of all the fault-free PEs on a particular time value and a rate of change of that value. Second, a reliable communication is one processor forming a consensus with another processor on some set of information and the order of transmission of that set [Birman and Joseph 1987]. Third, a fault-tolerant computation is the result of a consensus of the fault-free PEs on the fault-free status of the PE that performed the computation, or on the correctness of the computation itself. Finally, reconfiguration is a concurrence by all service users on the status of their servers. Therefore, at a high level, the fault-tolerant, distributed operating system of Fig. 1 consists entirely of consensus procedures.

Another general method of implementing fault tolerance is the state machine approach [Schneider 1990; Cristian 1991a]. Here, a fault-tolerant service is created by replicating the desired server and its service requests. At the heart of this technique is the coordination of the service population such that the failure of a member will be recognized and tolerated. In other words, the fault-free constituents must agree on who is faulty and what is the desired result. Thus, the basis of this approach is a consensus procedure yet again.

Traditionally, the formation of a consensus among several processors has been implemented with *n-modular redundancy* (NMR) at a great cost of resources while only attaining the throughput (jobs per unit time) of a single PE. With NMR, n PEs perform the same task. Thus, t faulty PEs, $n \geq 2t + 1$, may be *masked* by taking a majority vote of the n results.

The throughput of this n processor system could be increased by the number of fault-free PEs if one could reliably determine which of the PEs were faulty. Then, rather than mask the faulty processors, the system could identify and ignore them, thus allowing unique tasks to be scheduled on each fault-free processor, increasing the performance of the system over the NMR technique by the number of fault-free PEs. Therefore, a natural goal is to diagnose, i.e., detect and locate, faulty processors and to disseminate this information to the fault-free processors. If the diagnosis is correct, then the result of each processor is as reliable as the majority result of the NMR case. The field of *system diagnosis* has explored solutions of this type for over twenty years [Preparata *et al.* 1967], and the results are applicable to wafer scale integration, large, loosely-coupled, distributed computer networks and to other kinds of multicomputer systems [Rangarajan *et al.* 1990; Somani and Agarwal 1989; Kuhl and Reddy 1980]. Surveys on system diagnosis may be found in [Dahbura 1988; Friedman and Simoncini 1980; Kreutzer and Hakimi 1987; Kime 1986; Malek and Liu 1980].

A problem with diagnosis is that the fault status of the system is obsolete, although possibly correct, as soon as it is calculated. Most likely, a fault will require a recovery. Therefore, NMR techniques may still be needed when any recovery procedure would be too costly. But, implementation of NMR requires an ultra-reliable voting mechanism, accessible by the n replicated PEs, that is typically not available in a distributed system. Without this voter, all fault-free processors must be able to reach, by message passing, a consensus on some global datum, namely the correct result, despite the specious outputs of the faulty PEs. Work on the *Byzantine Generals Problem* (BGP) or *Byzantine agreement* explores this consensus problem. A survey may be found in [Raynal 1988].

Despite their different characteristics, solutions to the BGP and system diagnosis both

have very similar goals, namely to produce a correct result despite a number of faults. But the two areas have developed entirely apart with entirely different assumptions guiding their development. One goal of this paper is to show the similarities in purpose of the two approaches, and to allow future research to draw from both areas rather than to continue apart.

In this second section, the problems of system diagnosis and Byzantine agreement are discussed as they were originally presented. Included in this discussion are some of the immediate ramifications of these proposals. In later sections, the extensions and transformations that these early works underwent are outlined.

2.1 The PMC Model

A system operating in a tightly- or loosely-coupled, distributed environment must avoid giving tasks to or using results from faulty processing elements. Therefore, it is necessary for a centralized operating system, or for every processing element, to be aware of the condition of all the active PEs. In 1967, Preparata, Metze and Chien (PMC) formed the framework for much of the research in the system diagnosis area with their model of this problem [Preparata *et al.* 1967]. They eliminated the steep cost of NMR and special testing hardware by considering that a PE could test other PEs and that the results could be used to find the state of the system. However, test results may not be reliable if the testing PE is faulty!

The PMC model uses a graph $G(V, E)$ to model the system's testing convention. PEs make up the set V and directed edges in E represent one processor applying a test to another processor, i.e., the edge (A, B) denotes that A tests B. The edges are labeled with a 0(1)

if the test produces a passing(failing) result. The set of results is known as a *syndrome*.

After completion of testing according to G, a centralized arbiter interprets the syndrome and deems each PE to be either faulty or fault free. Certain assumptions are made about the faulty processors.

A1 All failures are *hard* or permanent faults.

A2 A fault-free processor is always able to determine accurately the condition of a PE it is testing.

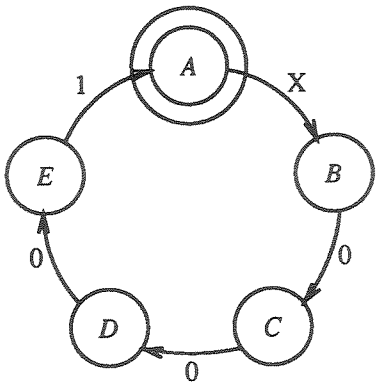
A3 A faulty processor produces unreliable test results.

A4 Not more than t PEs may be faulty.

These assumptions are not necessarily valid nor desired in a fault-tolerant, distributed network, and later work has dealt with removing these restrictions. The first problem with the PMC assumptions is supervisor-controlled diagnosis. The implication is that all test data must be gathered, analyzed and redistributed by a single PE. This is costly in terms of time, message-passing and system reliability, which is directly related to the reliability of the supervisor. Assumption *A1* disallows intermittent and transient faults. *A2* assumes that a test exists which is *complete* or has 100% fault *coverage*. In reality, the coverage will be less than 100% even for a simple PE. *A*4 may exclude many, practical, fault situations.

Fig. 2 shows a five processor system where A is faulty. The "X" on the edge (A, B) means that this result may be a one or a zero according to *A3*. If it is assumed in *A4* that $t = 1$, then it is possible to identify A as being faulty. First notice that edge (E, A) is labeled "1" meaning A is faulty if E is fault free. If E were faulty, then it would be the single faulty member of the system. So diagnosis depends on deducing the condition of E.

Either E is faulty or A is. Assume E is faulty in which case D must be fault free since $t = 1$. But this leads to a contradiction of *A2* because the label "0" on edge (D, E) implies that fault-free D misdiagnosed faulty E. Thus, E is fault free and A is faulty by *A2* regardless of the actual value of "X". If more than one PE is faulty, then credible system diagnosis is not feasible under this model [Preparata *et al.* 1967].



Tester	Tested	Result
FF	FF	0
FF	F	1
F	FF	X
F	F	X

Figure 2: Example of the PMC Model.

Preparata *et al.* were primarily interested in systems that allowed unambiguous diagnosis in all cases under assumptions *A1* through *A4*. Such systems are said to be *t-diagnosable*. In other situations, though, they considered diagnosis in conjunction with system repair. If it is not practical to diagnose a system in multiple phases, then it must be possible to identify all the faulty processors after one round of testing. In this case, diagnosis is called *one-step diagnosis* or *diagnosis without repair*. If the system is repairable, then it is only necessary to locate at least one faulty PE if it exists. In this case, after a PE is diagnosed as faulty, it can be repaired and the testing continued to eventually diagnose all the faulty PEs.

Such diagnosis is called *k-step diagnosis*, *sequential diagnosis*, or *diagnosis with repair*. For example, in Fig. 2 consider that A and B are both faulty. The syndrome is valid since tests given by both A and B are unreliable. With these two faulty PEs, it is not possible to determine with the given syndrome whether B is faulty. However, using our previous argument, A must be faulty. After repairing A, the condition of B is made obvious in the next round of testing.

Unless otherwise stated, "diagnosis" will refer to diagnosis without repair in the remainder of this paper.

Preparata *et al.* showed that if as many as t members of the PE population may be faulty according to $A4$, then it is necessary for the system to contain n members, $n \geq 2t+1$, to be diagnosable in all cases. Moreover, it is necessary that each PE be tested by at least t distinct other PEs. Hakimi and Amin showed that for the special case when no two processors test each other, these necessary conditions are also sufficient for *t-diagnosability* [Hakimi and Amin 1974]. Formally, a system is t-diagnosable if all faulty PEs may be uniquely identified, without repair, given the test syndrome, and provided that the number of faulty PEs does not exceed t [Preparata *et al.* 1967].

The *characterization problem* is to find necessary and sufficient conditions for a testing assignment to achieve a given level of diagnosability given a fault model and an allowable family of fault sets. Hakimi and Amin gave a general solution for one-step t-diagnosable networks [Hakimi and Amin 1974]. As before, $n \geq 2t+1$ and each PE must be tested by at least t distinct other PEs. But also, for each integer $p, 0 \leq p < t$, every subset X of processors, whose cardinality is equal to $n-2t+p$, must be tested by more than p processors outside of X. Huang *et al.* characterized sequentially t-diagnosable systems [Huang *et al.*

1989].

The *diagnosability problem* is to determine the family of fault sets that a given testing assignment can diagnose for some fault model. Sullivan solved the diagnosability problem given the PMC assumptions using *network flow* [Sullivan 1984; Deo 1974]. With his $O(|E|n^{1.5})$ algorithm, where E is the number of tests, it is possible to calculate the t-diagnosability of a given testing assignment. Recently, Raghavan and Tripathi improved the efficiency of the t-diagnosability algorithm to $O(nt^{2.5})$ [Raghavan and Tripathi 1991a]. They also showed that finding the diagnosability of repairable systems, i.e., sequential diagnosability, is co-NP-complete [Raghavan and Tripathi 1991b].

The *diagnosis problem* is to determine a fault set from a given family, for a given testing assignment, fault model, and syndrome. Fujiwara and Kinoshita showed that it is an NP-complete problem to find a set of minimal cardinality that, if faulty, could produce a given syndrome on a graph with arbitrary testing assignments [Fujiwara and Kinoshita 1978]. Thus, arbitration of conflicting test results is also NP-complete.

Still, work has been done on diagnosis in restricted situations. For t-diagnosable systems, Kameda, Toida and Allan (KTA) gave an $O(t|E|)$ algorithm, where $|E|$ is the number of tests, in which PEs are successively supposed to be faulty or fault free [Kameda *et al.* 1975]. This supposition and the test syndrome implicate the states of other PEs. If a contradiction occurs, the algorithm backtracks and tries again until it finds a consistent fault set. Recently, Sullivan improved the KTA solution to $O(t^3 + |E|)$, which is the best known solution when t is small ($O(n^{5/6})$) compared with n [Sullivan 1988].

Otherwise, the best solution in terms of worst-case efficiency is given by Dahbura and Masson [Dahbura and Masson 1984a]. They presented a $O(n^{2.5})$ algorithm in which an

undirected graph G is created whose vertices are the processors in the system and whose edges represent the *implied faulty sets* of each PE. The procedure is as follows: choose a PE and assume it is fault free. If this implies by the test syndrome that some PEs are faulty, then an edge should be drawn between the assumed fault-free PE and the implied faulty PEs. Note that self-loops might be produced. Repeat this for all the processors to create G. Then the faulty PEs are the *unique minimum vertex cover set* of G [Deo 1974], and by virtue of the class of graphs that must include G, these faulty processors are locatable in polynomial time. Dahbura and Masson gave a practical variation of their algorithm in [Dahbura and Masson 1984b].

Dahbura *et al.* studied the practical efficiency of the $O(n^{2.5})$ algorithm with respect to the KTA procedure and found that for small n ($n \leq 30$) the KTA method is almost always more efficient [Dahbura *et al.* 1985a]. Even for larger values of n, the KTA algorithm performs more efficiently on average than the method given by Dahbura and Masson. The KTA scheme guesses at a correct solution and backtracks if necessary. For an obvious fault syndrome, little or no backtracking is needed. But with the $O(n^{2.5})$ algorithm, a standard procedure must be executed for every fault situation and herein lies the discrepancy between the efficiency of the two approaches.

There are many special classes of t-diagnosable systems that support more efficient diagnosis techniques than those previously mentioned, and this is reason to believe that an $O(|E|)$ diagnosis solution exists for all t-diagnosable systems. Preparata *et al.* defined the $D_{\delta t}$ structure in which processor u_i tests u_j if and only if $j - i = \delta m$ (modulo n) where $m = 1, 2, \ldots, t$. They showed that if δ and n are relatively prime, then the system is one-step t-diagnosable [Preparata *et al.* 1967]. Meyer and Masson, Mallela, and Chwa and Hakimi

all gave $O(nt)$ solutions to the case of $\delta = 1$ [Meyer and Masson 1978; Mallela 1980; Chwa and Hakimi 1981b]. (Note that the characterization of t-diagnosable systems makes nt the minimum value of E [Hakimi and Amin 1974].) Maheshwari and Hakimi described the Z_T systems, Chwa and Hakimi gave the $D(n, t_0, X)$ class, and Dahbura *et al.* defined a group of "self-implicating" structures all of which have $O(|E|)$ diagnosis algorithms [Maheshwari and Hakimi 1976; Chwa and Hakimi 1981b; Dahbura *et al.* 1985b]. Sullivan developed an $O(|E|)$ algorithm for the most general class of test graphs among these mentioned, the t-vertex-connected digraphs which are a superset of the self-implicating structures given by Dahbura *et al.* [Sullivan 1984].

Researchers have refined and detailed the model given by Preparata *et al.* in search of more realistic assumptions and more practical solutions. Several of these extensions are examined later in the paper.

2.2 The Byzantine Generals Problem

The Byzantine Generals Problem is concerned with agreement among the fault-free segment of a population. The historical problem involves a group of Byzantine generals who have surrounded the enemy with their many armies. They wish to organize a concerted attack by sending messengers back and forth amongst themselves. The enemy is clever, though, and has been sending his own messengers with sometimes false and sometimes true messages to the Byzantine generals. The problem is to devise a scheme that will guarantee that the Byzantine generals agree to either attack or retreat.

The Byzantine generals are replaced by processing elements in a distributed computing environment. Every PE has a secret, binary value that it wishes to broadcast to every

other processing element. In a correct solution, all fault-free PEs should form identical vectors (*consistency*) whose elements corresponding to other fault-free PEs should be the secret values of those processors (*meaningfulness*). Together, these two conditions assure *interactive consistency* [Pease *et al.* 1980]. Faulty PEs may work in collusion to try to break the agreement by sending inconsistent information to different processors.

Pease *et al.* introduced and studied the Byzantine Generals Problem [Pease *et al.* 1980; Lamport *et al.* 1982]. They assumed that any two PEs have direct communication across a network that is not affected by the failure of connected processors, nor prone to failure itself, and has negligible delay. The sender of a message is identifiable by the receiver. With no other assumptions, including no centralized arbiter as in the PMC model, they showed that for *Byzantine agreement* to be reached, it is necessary that $n \geq 3t + 1$, where n is the number of PEs and t is the maximum number of those that may be faulty.

As an example, let $n = 4$ and $t = 1$. Byzantine agreement is reached after two rounds of message passing. In the first round, the processors exchange their private values. If a PE fails to receive an expected message, then it simply assigns a default value to that message. In the second round, the PEs exchange all of their information obtained from the first round. Now every processor has three numbers for the secret value of each other PE. Pease *et al.* showed that the majority (which will exist as the values are binary) among the three numbers for a particular PE is the secret value for that PE or the default value, and that all fault-free PEs will reach the same decision about the value held by each other PE.

Unlike the system diagnosis algorithms in which arbitration of conflicting test results is NP-complete except for special cases, this Byzantine agreement procedure can resolve conflicting values by simply taking a majority vote of the values received at each processor.

Not only is arbitration simple, but it is also completely distributed.

The algorithm for solving the Byzantine Generals Problem with n PEs involves the same sort of message passing as in the previous example and requires $t + 1$ rounds to complete (Fischer and Lynch showed that at least $t + 1$ rounds are needed for all deterministic solutions to the Byzantine Generals Problem [Fischer and Lynch 1982]) [Lamport *et al.* 1982]. Unfortunately, the message size grows exponentially with each round. Work on more efficient BGP algorithms may be found in Section 5.2.1.

There are a number of important points that need to be mentioned about the algorithm given by Pease *et al.*. First, the target system must be synchronous, i.e., a processor's failure to send a message is detectable. Without some sort of synchronization, Fischer *et al.* proved that Byzantine agreement is impossible even if only one processor crashes during the protocol [Fischer *et al.* 1985]. Second, the system is completely connected with private communication channels. Although this is unnecessary, Dolev showed that the connectivity of the communication graph must be at least $2t + 1$, and that reducing the connectivity will most likely result in more rounds required for Byzantine agreement [Dolev 1981; Dolev 1982]. Third, the secret values of the PEs are binary. Byzantine agreement protocols incur an extra cost if the secret values of the PEs each consist of k bits, although in the worst case one could simply iterate the algorithm k times to form a consensus on k bits. Obviously, this increases the cost of the algorithm by a factor of k [Turpin and Coan 1984]. Finally, messages are *unauthenticated*.

A message is *authenticated* if: 1) a message signed by a fault-free PE is unforgeable; 2) any corruption of the message is detectable; and 3) the signature can be authenticated by any other PE. Obviously, this limits the capabilities of the faulty processor. In this situa-

tion, there is no limit on the number of faulty processors that are tolerable and the network no longer requires private communication channels between PEs. (Actually, it is precisely private (point-to-point) links that provide the opportunity for inconsistency!) Trivially, the connectivity of the communication graph must be $t + 1$. Dolev and Reischuk gave an algorithm using authenticated messages that requires $t + 2$ rounds and $O(t^2)$ messages [Dolev and Reischuk 1985]. Algorithms using authentication have been less researched than their non-authenticated counterparts because of the demands placed on the processors and communication system by the authentication process. Until the efficiency of the authentication process is given, the true efficiency of an algorithm using authentication is unknown.

With a system diagnosis algorithm, every processor must pass a trial of tests. If it passes, then its output is assumed to be correct. In this way, every processor may operate on its own set of jobs. With a Byzantine agreement protocol, though, every processor is treated as if it were fault free, but enough processors are doing the same task that all faulty results may be masked out. At a high level, the two solutions are behaving identically. That is, a multiprocessor is given a set of inputs, and despite any failures, it is returning a correct set of outputs. The difference lies in the performance of the two algorithms. The system diagnosis solution should operate with a high throughput until a fault requires recovery, but the Byzantine agreement protocol should perform with extremely high reliability, and with a consistent, though lower, throughput. Simply, they are two similar algorithms with different performance characteristics.

In later sections, the evolution and extensions of the Byzantine agreement protocols are examined. First, though, the characteristics of a faulty processor are discussed.

3 THE FAULTY ELEMENT

Knowing how a processor element fails is key to making realistic assumptions and creating workable algorithms to detect and tolerate the faulty PE. Careful examination of the characteristics of faulty processors has resulted in the proposition of many fault models. The relevance of any model depends on the system in question, but in general, the more constraints in the fault model, the easier it will be to form a consensus.

3.1 Fault Models

Determining the interactions of faulty PEs is the essence of the consensus problem. For system diagnosis, these interactions are most pronounced during testing when test results have different possible interpretations given the assumptions about how processors fail. In the PMC model, a faulty PE performing a test on another PE will report unreliable results and a fault-free PE performing a test on another PE will always produce correct test results. This is known as *symmetric invalidation*. Fig. 3 shows some other proposed test result models. Barsi *et al.* introduced the BGM model in [Barsi *et al.* 1976]. Their assumption was that a faulty processor would always appear faulty regardless of the condition of the testing processor. Given a large number of test stimuli, it is reasonable that at least one set of expected and actual results will mismatch if the tested PE is faulty, even if the tester is faulty. This assumption is known as *asymmetric invalidation*. Hakimi and Kreutzer extended this assumption by proposing that a faulty tester would always report a non-faulty PE as being faulty [Kreutzer and Hakimi 1983]. The HK Model 1 and HK Model 2 are called *reflexive* and *irreflexive invalidation*, respectively.

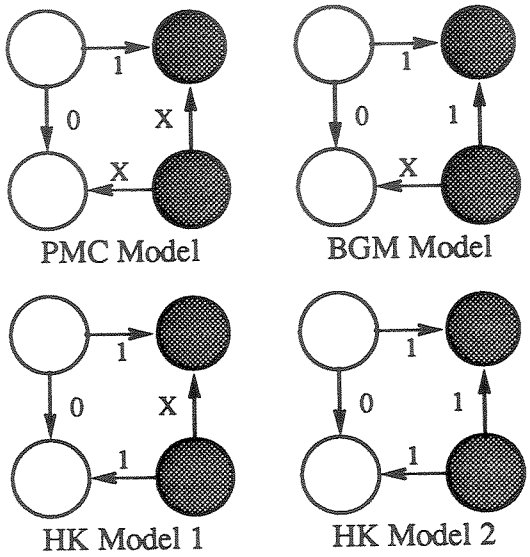


Figure 3: Test Validity Models. (A directed edge denotes a test by a PE on another PE. A 0(1) denotes a pass(fail). An X indicates that the PE will produce an unreliable result after performing the test. Faulty PEs are gray.)

Classically, solutions that reach Byzantine agreement make no assumptions about the characteristics of the faulty processor. In fact, faulty processor members are assumed, in the worst case, to work in collusion with complete knowledge about the state of the system. This *adversary model* is of course the safest and most conservative approach one could take to modeling a real system, but the lack of limitations means a defense will be expensive. Methods such as message authentication techniques or providing hardware broadcast mechanisms, i.e., a bus, do constrain the faulty PEs by imposing limits on their computational power or on their maliciousness. Of course, the adversary must be constrained to some extent. For example, the number of processors controlled by the adversary is limited so that it cannot simply cause every processor to fail immediately. Chor and Coan gave four principal handicaps to the adversary: 1) the adversary may corrupt fewer than one third of

the processors (see Section 2.2 for a description of this limit); 2) the communication system is reliable and unreliable links must be simulated by corrupting one of the two communicating processors; 3) the adversary may not predict random events; and, 4) the adversary must obey the synchrony of the system [Chor and Coan 1985]. As in system diagnosis, limiting the fault model simplifies the solution [Lamport *et al.* 1982].

3.2 Fault Classes

A diagnostic procedure must take into account the possible fault classes. Processor faults are categorized as *transient, intermittent*, or *permanent*. Transient faults are caused by events that come from a system's environment, and do not imply that the system is faulty. An intermittent or *soft* fault originates from inside the system when hardware is faulty. By its nature, an intermittent fault will not occur consistently, which makes its diagnosis a probabilistic event over time unless the fault becomes permanent or *hard*. Permanent faults are software or hardware faults that always produce errors when they are exercised [Johnson and Malek 1989].

It is difficult to determine the difference between a transient and an intermittent fault by simply observing the system. A fault caused by external events may have the same characteristics as one caused by internal events. The importance of the distinction is that the transient fault does not necessarily imply that the system should be declared faulty although the unstable environment might warrant a temporary shutdown. On the other hand, if the fault is intermittent, the system should be declared faulty until the problem is corrected. If it were assumed that only transients faults occurred, or that intermittent faults were very rare, then it would possibly be more productive to leave the affected PEs

in the processor pool, performing recovery procedures as necessary, than to remove, repair, and rejoin them.

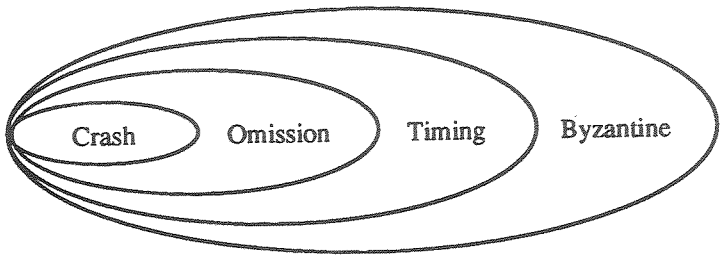


Figure 4: A Time Domain Fault Classification.

Cristian *et al.* classified processor faults into four major groups [Cristian *et al.* 1986]. These are *crash* faults, *omission* faults, *timing* faults, and *Byzantine* faults. Each class is a subset of the class that is listed next. Fig. 4 shows a graphical representation of this.

crash fault: The fault that occurs when a processor loses its internal state or halts. For example, a PE that has had the contents of its instruction pipeline corrupted, or has lost all power has suffered a *crash fault*.

omission fault: The fault that occurs when a processor fails to meet a deadline or begin a task.

timing fault: The fault that occurs when a processor completes a task either before or after its specified time frame. This is sometimes called a *performance fault*.

Byzantine fault: An arbitrary fault such as when one processor sends differing messages during a broadcast to its neighbors. More generally, this is every fault considered in the system model.

incorrect computation fault: The fault that occurs when a processor fails to produce the correct result in response to the correct inputs.

The incorrect computation fault class is a superset of the crash, omission, and timing fault classes and a subset of Byzantine failures. The first characteristic is true because a miscalculation may take place in time or space. Since the fault is consistent to all outside observers, though, the incorrect computation class is stricter than Byzantine faults [Laranjeira *et al.* 1991].

The most basic fault classes, crash, timing and performance failures, are problems that occur in the time domain, and are problems that are detectable in the time domain. This is in contrast to the more common fault classes mentioned previously that stress error detection in the data domain.

3.3 Fault Impact

The *impact* of a fault is the functionality reduction caused by that fault. A fault in one module of a system may or may not affect the operation of other modules. The impact of a fault on a PE will determine both if a particular test on that PE will declare it faulty, and if that PE can reliably perform a particular test. After a fault, a PE may stop communicating, start sending corrupted data, slow down its computations, stop performing some functions, begin performing functions incorrectly, or some combination of the above, that may or may not affect its ability to perform the tasks assigned to it.

4 THE TEST

After the faulty processing element has been characterized, the next step is to derive a test that will uncover it. This is the case for system diagnosis algorithms, and various techniques are discussed in this section. Byzantine agreement does not intentionally diagnose elements, and therefore, is not restricted by the limitations of a test.

The nature of tests in system diagnosis is a major point of contention in practical systems. Typically, processor A tests processor B by giving it certain inputs and comparing the resulting outputs with some set of correct responses. A quick and complete test is desired because without one, a faulty PE could go undiagnosed for an unacceptable period of time, or forever, and cause unrecoverable damage to the system state. It is obvious that a test cannot be allowed to tie up a normally busy PE with diagnostic tasks nor can it overload a congested network. Yet, for a highly complex system, a test could take hours or days and still not produce accurate results. This section looks at the test and the means of making it efficient.

4.1 Self-Testing

Testing may be performed by each processing element on itself in a series of *self-tests*. Thus, a direct test of processor A on B becomes a simple request for the status of B to which the self-checking mechanisms of B will respond. In this case, all free time at B may be spent testing without using the network. Kuhl and Reddy describe a hierarchical system of self-tests that permit a PE to deem itself faulty or fault free, including varying degrees of self-diagnosability, by means of hardware or software checkers, watchdog timers,

error-detecting codes, or redundancy with voting [Kuhl and Reddy 1980].

4.2 Group Testing

A test may only be able to determine whether a group of PEs is faulty or fault free, and reaching a single PE resolution might require multiple tests. Also, the execution of a test may require multiple units where failure of one of these units would invalidate the result [Kime 1978; Russell and Kime 1975a; Russell and Kime 1975b]. If many independent modules are required to perform a test, then the system is described as Multiple Invalidations Per Test (MIPT) as opposed to Single Invalidation Per Test (SIPT) that is the case in the PMC model. If a test has only multiple module resolution, i.e., the test cannot pinpoint a fault to a single module, then the model is referred to as Multiple Units Per Test (MUPT) as opposed to Single Unit Per Test (SUPT) that again is the case in the PMC model. The tests in a MIPT or MUPT environment might be simpler to write and quicker to execute, because less demands are made on them, and depending on the impact of a fault, the test might be sufficient. When tests fail, a table may be examined to determine what specific faults or group of faults could cause the test set to fail. A table also could be used to schedule the next round of tests to locate or avoid faulty units. Maheshwari and Hakimi characterized MIPT/MUPT systems while Holt and Smith examined their diagnosability and diagnosis [Maheshwari and Hakimi 1976; Holt and Smith 1981].

4.3 Comparison Testing

Typically, a test consists of performing an action and comparing the result of that action with that which is expected. If the result disagrees with the expected answer, then an

error has occurred. The problem with this approach is that typical electronic units are too complex to have such a test be able to determine unambiguously in a reasonable amount of time whether they are faulty or fault free.

A practical method of detecting faulty components is by comparison. Determination of the faulty or fault-free status of elements in the system is made by assigning a task to a pair of elements and comparing the results. When comparing results from two PEs one can detect, but not diagnose, a failure. When comparing results from more than two PEs, one can diagnose up to $\lceil n/2 \rceil - 1$ processors using NMR techniques. If the failure rate of two PEs is low, then it is not expected that they will fail at the same time, nor is it necessarily expected that they will fail in the same manner. Therefore, two similar PEs performing identical, deterministic tasks should produce identical results unless one, or even both, of them have failed. The comparison method is not foolproof, though. An intermittently faulty PE could produce correct results for certain test tasks, or two faulty PEs could report the same incorrect results. Nonetheless, the comparison technique promises high fault coverage with detection in a short amount of time [Rangarajan *et al.* 1990].

Whereas the tests of the PMC model are performed in rounds between system tasks, comparison tests can occur in conjunction with productive tasks. A fault is detected when it happens and allows maximum fault containment much in the same way as Byzantine agreement algorithms which also use a form of comparison. The comparison of results can be implemented by creating and comparing *signatures*, such as checksums or cyclic redundancy codes, of the results.

Malek introduced the comparison approach in the context of system diagnosis and presented a method to assign comparison edges in the system graph [Malek 1980]. A syndrome

of comparison results is created and diagnosed by a centralized supervisor. Chwa and Hakimi proposed a similar comparison approach independently [Chwa and Hakimi 1981a]. Maeng and Malek broached the problem of decentralizing the arbitration of comparisons by considering the use of a third processor to compare the results of two other processors [Maeng and Malek 1981]. The Maeng/Malek fault model is given in Table 1 where a 0 means the processors agreed, a 1 implies that they did not, and an X is an unpredictable result.

Table 1: Fault Model for Maeng/Malek Comparisons.

Comparator	Compared 1	Compared 2	Result
fault free	fault free	fault free	0
fault free	fault free	faulty	1
fault free	faulty	fault free	1
fault free	faulty	faulty	1
faulty	fault free	fault free	X
faulty	fault free	faulty	X
faulty	faulty	fault free	X
faulty	faulty	faulty	X

As an example, consider the four processor system of Fig. 5 in which A is faulty. (Note that, in this case, at most one PE can be faulty for diagnosis without repair.) Each PE is performing the same task for comparison purposes. When a processor completes its copy of the task, the result is broadcast to the other processors. After all tasks are completed,

each processor will have four values including its own. For example, say that each has received values $10, 24, 24, 24$ from A, B, C, and D respectively. The resulting syndrome, which each fault-free PE can construct, is shown in Fig. 5. The next step is to analyze this syndrome, for example, using the $O(n^{2.5})$ algorithm of Dahbura and Masson described in Section 2.1 [Dahbura and Masson 1984a]. A graph G is created with the same processors as in the system. Assuming A is fault free implies that B, C, and D are faulty, so the edges $(A, B), (A, C)$, and (A, D) are added. Assuming B, C, and D are faulty adds no new edges to the system. Then, the minimum vertex cover set of G is A since all edges have one end at A. Therefore, A is the faulty processor.

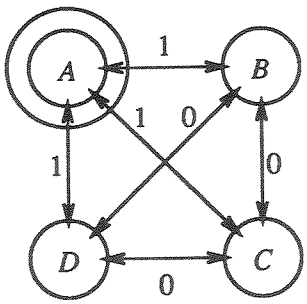


Figure 5: An Example of Comparison Testing and Diagnosis.

Finding the complete and correct set of faulty processors using the comparison model is NP-complete, but if the system is t-diagnosable, the problem is solvable in polynomial time [Sengupta and Dahbura 1989]. Sengupta and Dahbura characterized the comparison assignments of the system and gave an algorithm using the Maeng/Malek model.

4.4 Time Domain Testing

Cristian models processor faults in the time domain, Fig. 4, and processors are tested with respect to time [Cristian 1991a]. If a PE fails to complete a task, or send or receive a

message within some time frame, then an error has occurred. Timing faults can be detected with simple tests using timestamps and time-outs in the case of a global set of synchronized clocks. Moreover, this time-domain fault-model is orthogonal to the data domain techniques previously described.

Most early system diagnosis research did not concentrate on the limitations of the test. Instead, it was assumed that a test was available with whatever requirements were needed. In the next section, though, much work is presented that weakens the classic assumption of 100% coverage. The explicit details of the test are ignored, yet it is understood that any implementation of the test will be imperfect.

5 SPECIFYING THE CONSENSUS PROBLEM

Research on the consensus problem has focused on specification. That is, the assumptions associated with the problem have been strengthened or weakened dependent on the specific system which is to support the consensus protocol. This section looks at extensions given to the basic system diagnosis model, the PMC model, and to the Byzantine agreement algorithm given by Pease *et al.*.

5.1 Extensions to the PMC Model

The original system diagnosis model and diagnosis goals set forth by Preparata *et al.* made a number of stringent demands on the underlying hardware [Preparata *et al.* 1967]. As a result, system level diagnosis has had a limited impact on fault-tolerant system design. Dahbura gave several, simplifying assumptions that have guided much research in the area and which need to be examined to change this situation [Dahbura 1988].

Test scheduling. Often, diagnosis is considered the only task of the system, and the effect of operating in conjunction with other tasks is ignored. But in fact, it is desirable that test scheduling minimally impact the throughput of the system without diagnosis. One way of doing this is to use the *spare capacity*, or temporarily unused resources, of the multiprocessor system to perform testing and analysis. For moderately loaded systems, a sufficient percentage of jobs may be duplicated in the spare capacity to provide a basis for fault detection and diagnosis with virtually no degradation to system response time [Dahbura *et al.* 1989]. *Roving diagnosis* introduced by Nair *et al.* uses a time-varying subset of PEs to perform the system tasks while the other PEs conduct testing and diagnosis [Friedman and Simoncini 1980]. Concurrent and adaptive diagnosis techniques strive to reduce the effect of testing and analysis on system performance.

Worst-case approach to diagnosis. Typically, fault diagnosis algorithms are designed to identify the fault set under all circumstances including such improbable cases as faulty PEs colluding to diagnose fault-free PEs as faulty and vice versa. More efficient algorithms can be developed if these situations are ignored, or if the straightforward solution is a priority of the strategy. The effect of this was discussed in Section 2.1 and studied in [Dahbura *et al.* 1985a].

Hardware faults. System diagnosis has been intended primarily for treating hardware faults as opposed to design flaws in software or operator errors. The redundancy management aspects of system diagnosis are surely applicable, but the hardware/software analogy has not been carried through fully. For example, the implications of testing a piece of software or user inputs need examining. It is uncertain whether hardware or software faults will predominate future multiprocessor systems, and it is uncertain what role system diagnosis

will play in the latter situation.

Centralized diagnosis. Often it has been assumed that an ultrareliable, supervising arbiter is available to analyze test syndromes, and disseminate diagnosis information. The implementation of such a device would place a bottleneck on performance, reduce availability, and impair expandability. For these reasons, *distributed diagnosis* has been introduced and studied [Kuhl and Reddy 1980; Kuhl and Reddy 1981; Hosseini *et al.* 1985].

The relaxation of these simplifying assumptions is the focus of the next several sections. Together they represent the current status of system-level diagnosis and, in many ways, a new approach to fault-tolerant system design.

5.1.1 Set Diagnosis

Friedman proposed that replacing a set of processors, including some that could be fault free, might be acceptable when single processor diagnosability is not practical. He called a system t/s-diagnosable if the set of at most t faulty PEs are identifiable to within a set of at most s PEs [Friedman 1975]. Karunanithi and Friedman looked at the effect of t/s-diagnosability on the diagnosis of certain network topologies [Karunanithi and Friedman 1977].

A special and important case of set diagnosis is t_1/t_1-diagnosis that was characterized by Chwa and Hakimi [Chwa and Hakimi 1981b]. A system might be t-diagnosable and t_1/t_1-diagnosable with $t \leq t_1$. If $f \leq t$, where f is the number of faults, then the fault set is obviously identifiable. If $t < f \leq t_1$ then Yang *et al.* showed that all the faulty PEs except at most one could be correctly identified and isolated in a set of cardinality less than or equal to t_1, which will contain, at most, one fault-free element [Yang *et al.* 1986].

In addition, the status of each PE in the set can be determined to be either "faulty" or "unknown". The importance of this class of set diagnosis was shown by Kavianpour and Friedman who noted that if $n \gg s$, then only $n\lceil(s+1)/2\rceil$ tests are needed to construct a t/s-diagnosable system [Kavianpour and Friedman 1978]. This is almost half the number of tests required by a t-diagnosable system [Hakimi and Amin 1974].

Using conventional testing techniques, Yang *et al.* generalized the $O(n^{2.5})$ algorithm of Dahbura and Masson to achieve the diagnosis described for t_1/t_1-diagnosable systems [Dahbura and Masson 1984a; Yang *et al.* 1986]. Using the comparison approach, Yang and Masson generalized the backtracking algorithm of Chwa and Hakimi, similar to the KTA method given in Section 2.1, for $O(|E|)$ diagnosis where $|E|$ is the number of tests that were given above [Chwa and Hakimi 1981a ; Yang and Masson 1986].

Kavianpour and Friedman, and later Chwa and Hakimi examined the $D(n, t_0, X)$ class of systems which are t_1/t_1-diagnosable where t_1 may be much greater that t_0 [Kavianpour and Friedman 1978; Chwa and Hakimi 1981b]. A system is a $D(n, t_0, X)$ system if for a positive integer t_0, $t_0 \leq \lfloor(n-1)/2\rfloor$, and a set of integers X, $1 \leq x_1 < x_2 < \cdots < x_{t_0} \leq \lfloor(n-1)/2\rfloor$, an edge exists between PEs i and j if and only if $(i-j)$ (modulo n) $\in X$. Maxemchuk and Dahbura showed the optimal design of such systems reach $(2t_0-1)/(2t_0-1)$-diagnosability [Maxemchuk and Dahbura 1986].

Another special case of t/s-diagnosability is $t/(n-1)$-diagnosability which guarantees the location of one fault-free processor. This processor may be used in reliably testing other PEs as in adaptive testing (see Section 5.1.2) or it may be used to select the correct result from n PEs performing the same task. Xu examined $t/(n-1)$-diagnosability and its use in the diagnosis and repair of constant degree systems as well as software fault tolerance [Xu

1991]. Optimal configurations for these systems were presented.

5.1.2 Adaptive Testing

Diagnosis is performed with or without repair. Without repair, all testing occurs in a single round after which all diagnosis decisions must be made. With repair, though, a number of rounds will pass before diagnosis is completed. Nakajima saw that tests could be adapted as information was uncovered to optimize the speed and accuracy of the diagnostic process [Nakajima 1981]. Whereas the testing assignment of a system is typically determined before diagnosis, Nakajima proposed to determine this assignment dynamically. Blecher showed that, in the worst case, at least $n + t - 1$ tests are required for $n \geq 3$ [Blecher 1983]. An adaptive testing algorithm given by Hakimi and Nakajima first uses testing with repair to locate a fault-free processor. This processor then reliably tests (given that the processor does not subsequently fail) all other PEs in the system for efficient diagnosis [Hakimi and Nakajima 1984]. The parallelization of this basic algorithm, via broadcast operations, can diagnose a system in $O(\log_{\lfloor n/t \rfloor} t)$ rounds with $O(n)$ tests [Schmeichel *et al.* 1988].

5.1.3 Intermittent Faults

Mallela and Masson were the first to included intermittent faults in their system model [Mallela and Masson 1978]. This fault class adds complexity to the PMC model because it can no longer be assumed that a fault-free tester will accurately judge the condition of the PE that it is testing. So while all PEs that give faulty outputs are indeed faulty, other defective PEs might go undiagnosed, thus leaving the diagnosis *incomplete*. The solution is repeated testing until a test overlaps the occurrence of an intermittent fault. After a test

is failed, though, it no longer needs to be repeated as the fault has been uncovered. After every round of testing, a *subsyndrome* is produced. This subsyndrome is a subset of the actual syndrome which is the testing result that would be produced if every faulty PE were permanently faulty. Thus, a syndrome is *pf-compatible* or permanent-fault compatible as it could be produced by a system suffering only from permanent faults. A problem arises when a subsyndrome not equal to the system syndrome is also pf-compatible because its analysis could lead to an incorrect diagnosis of the system. Mallela and Masson characterized t_i-*diagnosable* systems, where t_i is the maximum number of intermittently faulty PEs. These systems will never produce a pf-compatible subsyndrome leading to an incorrect diagnosis. Thus, diagnosis is correct whenever a subsyndrome is pf-compatible, although it may not be complete, i.e., a faulty PE might go undiagnosed, but a fault-free PE will never be labeled faulty. They found that these systems have significantly more restrictive requirements than systems that are only t-diagnosable for permanent faults.

The t_i-diagnosability measure fails to account for PEs that exhibit hard-failure semantics, and that omission could hamper analysis. This *hybrid* fault situation is modeled by a t_h/t_{hi}-diagnosable system in which at most t_h PEs are faulty and of these at most t_{hi} are intermittently faulty [Mallela and Masson 1980]. Hybrid fault systems were detailed further as $t_h/t_{hi}/t_{hi}$-diagnosable if all the permanent faults in the corresponding t_h/t_{hi}-diagnosable system could be located [Yang and Masson 1985b]. Unfortunately, a system of this type requires a large number of testing assignments, but a procedure has been given for designing $t_h/t_{hi}/t_{hi}$-diagnosable systems [Kohda and Abiru 1988]. $t_h/t_{hi}/\tau$-diagnosability was introduced as the *master diagnosability measure*, so-called because it includes all the previous hybrid fault diagnosability measures as special cases [Yang and Masson 1987]. Two types

of intermittent failures are identified: *almost-hard* failures that occur with great enough frequency that a few tests will uncover them, and *very-soft* failures that are elusive to detection. Then t_h and t_{hi} are the same values as previously defined, and τ is the bound on very-soft failures. The authors have characterized all the systems described by these diagnosability measures.

The set of all test syndromes produced by a system that suffers from intermittent faults is a superset of all test syndromes possible in the same system suffering only from permanent faults. That is, some syndromes will not be pf-compatible. The implication of this is that previous diagnosis algorithms are no longer directly applicable. Dahbura and Masson introduced the idea of *greedy diagnosis* to identify faulty processors from a pf-incompatible syndrome [Dahbura and Masson 1983a]. They also applied this approach to comparison-based systems in [Dahbura and Masson 1983b]. The algorithm requires that a bound be put on the number of soft-failing PEs, and that the number of simultaneously failing PEs also be bounded. Unfortunately, this approach does not lead to a polynomial-time diagnosis algorithm in the most general case.

Yang and Masson reported an algorithm that correctly identifies all faulty PEs if the syndrome is pf-compatible, and at least one faulty PE in many cases where the comparison syndrome is not pf-compatible with $O(|E|)$ efficiency ($|E|$ is the number of tests required) [Yang and Masson 1985a].

5.1.4 Probabilistic Diagnosis

The processing elements of a system are not necessarily homogeneous nor operating under similar conditions. Therefore, the probability that one PE will fail in a given amount of time

is not equal to the same failure probability of another PE. Considering this in the fault model can make diagnosis more practical and more efficient. Techniques which assign probabilities to the correctness of a test or to the reliability of a processing element fall into the area of probabilistic diagnosis. It should be noted that an intermittent fault could be modeled by a test with imperfect detection characteristics, or by assigning a reliability to the faulty PE that corresponds to the probability that a test will detect the intermittently faulty PE. Thus, probabilistic diagnosis is well suited for systems that experience intermittent faults.

Maheshwari and Hakimi assigned a reliability to each PE in the network [Maheshwari and Hakimi 1976]. The reliability is simply the probability of a fault occurring in a given PE. They defined a probabilistically t-diagnosable (p-t-diagnosable) system as having, for every allowable syndrome, a unique, consistent, fault set whose probability of occurrence is greater than p. They gave necessary and sufficient conditions for these systems, and Dahbura generalized the $O(n^{2.5})$ diagnosis algorithm, see Section 2.1, for use with p-t-diagnosable networks [Dahbura and Masson 1984a; Dahbura 1986].

Blount took a different approach to probabilistic diagnosis by assigning a probability of correctness to each test rather than to the PEs themselves [Blount 1977]. Unlike Preparata *et al.* who assumed that tests had perfect coverage, Blount assigned a probability to each test, based on the conditions of the tester and the tested PEs, to specify the coverage. Procedures were given for determining the probability of correct diagnosis for a particular fault set, and for the entire system.

The general problem is to diagnose a system that suffers from intermittent failure and that has tests with imperfect coverage. Dahbura, Sabnani and King were the first to examine this problem using probabilistic diagnosis under the comparison approach [Dahbura *et al.*

1987]. They gave a simple diagnosis algorithm that diagnoses the system correctly with an extremely high probability with $O(n^2)$ operations. This system model avoided many of the pitfalls introduced by Preparata *et al.* including the need for complete tests, the permanent nature of faults, off-line testing, and an upper bound on the number of simultaneously faulty PEs.

Blough, Sullivan and Masson reexamined the problem based on the more conventional approach of one PE testing another [Blough *et al.* 1988]. They assigned a reliability to the processing elements, and a coverage probability to the tests. It was shown that performing correct diagnosis with a probability approaching one is impossible with fewer than $O(n \log n)$ tests, where n is the number of PEs in the system. Blough *et al.* gave an $O(|E|)$ solution where $|E|$ is the number of tests and is at least $\omega(n)n \log n$ where $\omega(n)$ approaches infinity (arbitrarily slowly) as n approaches infinity.

Blough *et al.* continued their work to reduce the number of required tests which in turn reduces the communication overhead of diagnosis and the need for physical communication paths. It was shown that probabilistic diagnosis is almost surely correct in a system sparsely and randomly connected by its test assignments [Blough *et al.* 1989; Scheinerman 1987].

5.1.5 Distributed Diagnosis

One drawback of the PMC model is that a centralized arbiter must gather and analyze the global test syndrome to diagnose the system. This dedicated processing unit or specialized hardware must not only be ultrareliable, but it must also have guaranteed communication links to all the members of the network. This function is difficult and expensive to implement in a truly distributed system, and is a weak spot in a fault-tolerant design.

Therefore, methods for distributed diagnosis have been developed in which every processor decides independently what is the fault-free population. Thus, as long as the bounds for diagnosability are met, the hardware to perform the diagnosis is available.

Of course, one problem with removing the centralized supervisor is relaying the diagnosis information to the system user. If the user is unable to test a processor, then it is a problem to decide from which PE to take the system diagnosis information as any of them might be faulty. Kreutzer and Hakimi discussed this quandary in [Kreutzer and Hakimi 1988]. Basically, they sought the minimal connections required between a centralized observer, that is, the user, and a t-diagnosable system in which every fault-free processor has the correct diagnosis of the system. They found that if authenticated messages, or message passing, were available, then only $t + 1$ PEs needed to be queried to learn the correct diagnosis. Without authenticated messages, $2t + 1$ PEs needed to be probed.

Smith gave simple system diagnosis algorithms that do not use a centralized observer and can be applied regardless of the system structure, but he did not describe the manner in which the test data would be distributed nor did he couch his discussion in a distributed system framework [Smith 1979].

Simoncini *et al.* saw the problem as distributing the centralized analysis of the syndrome [Ciompi *et al.* 1981]. In their MuTeam approach, testing was carried out as required for t-diagnosability, but once completed, results were not sent to a centralized observer, but rather were disseminated using a consensus protocol similar to Byzantine agreement. Then, each PE could calculate the system diagnosis from these results. The problem with this approach is that, while it assures that all processors have a consistent view of the diagnosis of the system, the syndrome dissemination is expensive and halts useful processing. Therefore,

techniques resulting in dynamic diagnosis have been explored.

Kuhl and Reddy introduced *distributed diagnosis* in [Kuhl and Reddy 1980]. A processor in a distributed environment has reliable information about only those PEs in its neighborhood, i.e., those that it can communicate with via direct communication paths. Data about the rest of the system is indirectly available from PEs outside of the neighborhood. *Distributed fault tolerance* is the notion that each fault-free PE should be able to independently and correctly diagnose the entire system, then use this knowledge to refrain from dealing with any elements deemed to be faulty, and to initiate fault recovery techniques [Kim and Yang 1986]. The authors assumed that fault-free PEs could accurately test any other PE, and that faulty PEs conducted tests with unreliable results.

Kuhl and Reddy, joined later by Hosseini, presented a series of **SELF** algorithms in [Kuhl and Reddy 1980; Kuhl and Reddy 1981; Hosseini *et al.* 1985]. **SELF2**, which meets distributed fault tolerance as described above, is outlined here. It is assumed that faults are permanent. Each processor P_i calculates a *fault vector* F_i whose j^{th} element is a 0(1) if P_i concludes that P_j is fault free (faulty). A processor P tests each of its neighbors and completes part of the fault vector. If a neighbor is faulty then this condition is broadcast to all the fault-free PEs that themselves test P (obviously P has a direct connection to each of these PEs). Whenever P receives a diagnostic message about some faulty processor Q that was previously considered fault free, it first checks that it believes the last relay PE of the message is fault free. P tests this sender again and if it passes, the information about Q is saved in the fault vector and the message is forwarded to all those fault-free PEs that test P. Otherwise, the diagnostic message is ignored, the sender is marked as faulty, and this information is sent to the testers of P [Kuhl and Reddy 1980].

Bagchi and Hakimi gave an optimal algorithm for the system model of Kuhl and Reddy that assumes no more than t faults and fault-free communication links [Bagchi and Hakimi 1991]. Their algorithm requires at most $n-1+p(t+1)$ diagnosis operations and $3n\log p+O(n+pt)$ messages from fault-free PEs where p is the number of fault-free PEs.

In **SELF3**, Kuhl and Reddy extended the **SELF2** algorithm to cover message corruptions caused by faults in the communication paths or by relaying a message through a faulty processor. They also weakened the necessary condition that the network have a connectivity of t to be t-self-diagnosable [Kuhl and Reddy 1981; Deo 1974]. In the **Modified Algorithm SELF3**, Hosseini *et al.* altered **SELF3** such that fault-free PEs are never temporarily misdiagnosed as faulty [Hosseini *et al.* 1985]. See also the **NEW_SELF** algorithm described in [Hosseini *et al.* 1984].

The work of Hosseini *et al.* illustrates the trend towards more reliable accounting of the nuances of an actual system. Liaw *et al.* modeled a heterogeneous distributed system with a graph theoretical model where processors are marked as *testing* or *non-testing* units [Liaw *et al.* 1982]. A testing unit is a processing element with the capability to test at least one other processor. A non-testing unit does not test any objects and relies on other PEs to give it diagnosis information. Communication links are categorized and may handle general communications, testing communications, or both. Distributed diagnosis procedures are given to specify between failed PEs and links. Hosseini *et al.* used these ideas in their algorithm for non-homogeneous distributed systems [Hosseini *et al.* 1985]. The problem of link failures also may be examined as a routing problem. A discussion of this may be found in [Bertsekas and Gallager 1987].

Yang and Masson considered the distributed diagnosis of a t_i-diagnosable system [Yang

and Masson 1988]. The soft-fail model they employed covers intermittent faults in both the PEs and links as long as the total number of faults does not exceed t_i. Because a faulty PE or faulty communication link does not necessarily produce errors at any particular time it is being exercised, the system may act in a very capricious way, and in a Byzantine manner. Therefore, the diagnosis is not and cannot be guaranteed complete. A set of maliciously faulty processors could postpone diagnosis indefinitely.

Fussell and Rangarajan used probabilistic diagnosis in a distributed environment to assure, with high probability, correct diagnosis in a system with arbitrary connectivity [Rangarajan and Fussell 1988; Fussell and Rangarajan 1989]. This result is similar to that of Blough *et al.* who showed that correct diagnosis could be attained in systems with constant connectivity [Blough *et al.* 1989]. (Berman and Pelc used the probabilistic model of Blough *et al.* in a distributed diagnosis scheme [Berman and Pelc 1990].) Using their diagnosis scheme, which requires only two testers per processor, Fussell and Rangarajan showed that reliable diagnosis was available for such minimal networks as rings.

Fussell and Rangarajan accomplished the reliable diagnosis of sparsely connected networks by comparing processor results across several tasks [Fussell and Rangarajan 1989]. They exploited the fact that a failed processor might not cause an error in every task it completes, and therefore, its results may still be used to test other processors. Since many tests are performed on each PE, many syndromes are created, none of which must match any of the other syndromes. This *multiple syndrome* diagnosis will be correct with a very high probability if the number of *tests*, not unique testing processors, of each processor grows as $\log n$. Therefore, their technique is applicable to arbitrarily connected networks. A more efficient algorithm was given by Lee and Shin [Lee and Shin 1990]. Similar work

based on directed testing and diagnosis with repair has given equally promising results for low, constant-degree systems [Blough and Pelc 1990].

Realizing that the testing scheme should not be limited by the PE with the lowest connectivity in the system, Rangarajan and Fussell adapted their algorithm to tailor itself to any system topology [Rangarajan and Fussell 1991]. Previously, the number of testers was set at two for each PE, but in fact the number of testers is variable. The method they gave requires only that the product of *the number of tests conducted on each processor by one of its testers* $\times$ *the number of such testers* grows as $O(\log n)$. Thus, the diagnosis algorithm may be adjusted at each PE as desired and limited only by the network topology at that PE.

5.1.6 Processor Membership

Cristian examined a problem similar to distributed diagnosis that he called *processor membership* [Cristian 1991b]. A processor is in the membership, or fault free, if it can maintain a timely schedule of "present" messages. The problem is to keep all the fault-free PEs informed of the membership regardless of whether PEs are joining or leaving the system. The solutions were given as an overlay of a synchronous system with *atomic broadcasts* which were described in [Cristian *et al.* 1986]. The atomic broadcast assures that all or none of the fault-free processors will receive the message (*atomicity*), that every receiver gets messages in the order that they are sent (*order*), and that the broadcast is completed in some known time bound Δ (*termination*). With this mechanism, Cristian presented three protocols for processor membership: the *periodic broadcast* protocol, the *attendance list* protocol, and the *neighbor surveillance* protocol [Cristian 1991b]. These protocols handle the cases of

faulty processors leaving the membership and fault-free or repaired processors joining the membership.

Cristian considered processor faults only in the time domain and at the message passing level. Data domain faults may be corrected or at least detected at a lower level using coding techniques. To detect timing faults, it is assumed that each processing element has a local clock that is synchronized to within a constant of every other local clock. A processor is faulty only if it fails to send an expected message during an expected time.

The *periodic broadcast* protocol requires that every processor broadcast a "present" message with some predetermined and globally known frequency. A membership begins when a processor broadcasts a "new-group" message at time T that will be received by all the fault-free PEs within $T + \Delta$ by the nature of the atomic broadcast. In response, each good processor broadcasts a "present" message. Therefore, at $T + 2\Delta$, every fault-free PE knows the membership of the system. This implies that the delay to join the membership is 2Δ. New rounds of "present" messages are scheduled to occur at $T+2\Delta+k\Pi, k = 1, 2, 3, \ldots$ where Π is some time interval based on the reliability of the system and the frequency of testing that is desired. If a PE falls out of the membership, then this will be detected when it fails to send a "present" message. The worst case is if this PE just initiated a "new-group" message, then at most $\Delta + \Pi$ time units will pass before it is detected [Cristian 1991b].

Consider the system in Fig. 6 in which A has become faulty. Assume that the ring is a broadcast bus and that δ is the broadcast delay, i.e., $\delta = \Delta$. Initially, A was not faulty, and all PEs were in the membership. Every Π time units, each processor would broadcast its "present" message on the bus, and thus the membership was maintained. A fails. The failure of A will be detected at δ after the next scheduled round of broadcasts, and at the

same time, all fault-free processors will have the new membership. Eventually, a repaired A may return by broadcasting a "new-group" message.

An obvious problem with the periodic broadcast protocol is that it fills the communication network with "present" signals at every testing round. This congestion is reduced for point-to-point networks by the *attendance list* and *neighbor surveillance* protocols at the expense of the system response time to faulty PEs leaving the membership. Thus, a highly volatile membership could be inappropriate for these techniques. In the attendance list protocol, one PE is assigned the task of periodically initiating a roll call that circulates around a logical cycle through the membership. Each PE must timestamp the attendance list and forward it. If the completed list does not return to the originator within a specified time frame then an error has occurred and a "new-group" request is made. The result is that the message overhead is decreased compared with the periodic broadcast protocol while maintaining the maximum time to join the membership and increasing the maximum time to detect a departure. The maximum departure detection time is proportional to the time it takes the attendance list to circulate through the membership [Cristian 1991b].

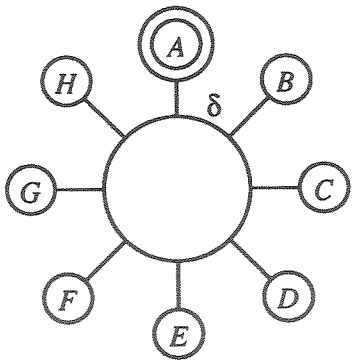


Figure 6: A Processor Membership Example.

Again consider Fig. 6, but suppose that the network uses point-to-point communica-

tion with δ being the delay on any direct link. Thus, Δ is $\lceil n/2 \rceil \delta$ making a broadcast a significantly more costly operation than in the previous example. To overcome this cost, the attendance list protocol is used. Initially, A is fault free, and an election algorithm has chosen A to periodically initiate the roll call. Every Π time units, n messages are sent, taking $n\delta$ time, as the attendance list is passed through the membership. A fails. B does not receive the list at the next scheduled round, and so broadcasts a "new-group" message which results in the formation of the new membership.

The neighbor surveillance protocol works in a manner similar to the attendance list protocol. A logical cycle of the processors in the membership is specified and given a direction. Periodically, each processor requests a *neighbor confirmation* of its predecessor. If the confirmation is not received during the correct time frame then a failure has occurred and a "new-group" request is initiated to establish the new membership. In the case of a single member departure, the worst case detection delay is better than the attendance list protocol since all neighbor confirmation messages may occur in parallel. The worst case detection delay is worse, though, in the case of multiple member departures [Cristian 1991b].

The processor membership problem is equivalent to the system diagnosis problem in that both strive to determine which processors are faulty and which are not. A test in the processor membership model is passed if a PE can transmit and forward messages in a timely manner. Therefore, these protocols will not withstand the malicious adversaries of the Byzantine Generals Problem, but they are adaptable to the work of Kuhl and Reddy. Recall that Kuhl and Reddy suggested processing elements should be equipped with self-diagnosis mechanisms that allow a test to be as simple as querying another PE for its status

[Kuhl and Reddy 1980]. Assume these checkers were available on a system using processor membership protocols. If a member has determined that it is faulty then it should refrain from broadcasting any "present" messages.

Processor membership does have important differences with distributed diagnosis. First, the protocols are closely related to time which means that they will detect timing failures and that they are well suited for real-time deadlines. Second, Cristian recognized the high cost of system overhead. He showed how detection time could be traded off for lower cost protocols. Finally, processor membership leads to a *consistent* view of the system, that is, every fault-free PE shares the same view of the system status with every other fault-free PE at all times. For certain applications, this might be a necessity.

5.2 Research on the Byzantine Generals Problem

The original Byzantine agreement algorithm presented by Pease *et al.* was expensive in both its communication and system requirements [Pease *et al.* 1980]. Therefore, two areas of work have emerged: efficient Byzantine agreement algorithms, and necessary system requirements for Byzantine agreement. In this section, these areas will be examined.

5.2.1 Efficient Byzantine Agreement

The algorithm given by Pease *et al.* requires $n = 3t + 1$ processors, $t + 1$ rounds and messages of the size $O(n^{t+1})$, as the amount of information exchanged between any two processors increases exponentially [Pease *et al.* 1980]. For Byzantine agreement, there are three independent resources: processors, rounds, and message size [Coan 1988]. An ideal, deterministic, unauthenticated Byzantine agreement procedure would use $3t+1$ processors,

$t+1$ rounds and messages of size one, and though it is possible to create procedures that are optimal in some of these respects, no algorithm optimized in all three categories has been found. For example, Coan gave an algorithm that uses $O(t^{1.5})$ processors, $t+1$ rounds of communication, and messages of size $O(t \log n)$ while Toueg *et al.* presented a scheme with $3t+1$ processors, $2t+1$ rounds, and message sizes polynomial in the number of processors [Coan 1988; Toueg *et al.* 1987].

Bar-Noy and Dolev asked if there even exists an algorithm that optimizes all three parameters [Bar-Noy and Dolev 1991]. They suggested the following problem as a reduction of this question: Is there an algorithm with one-bit messages that terminates after $t+1$ rounds with $n = O(t)$? An algorithm using $(2t+1)(t+1)$ processors, $t+1$ rounds, and one-bit messages, and which did not need to know where messages originated, was presented. One of the most exciting aspects of the algorithm is the ease with which it could be implemented in hardware. This is because: 1) messages contain only one bit of information; 2) rounds in which processors receive or send messages rely only upon the clock and are not data dependent; and 3) processors need only a simple piece of circuitry to calculate the majority result of a number of binary values to determine the message they should send.

Randomized Byzantine agreement. Randomized Byzantine agreement algorithms have been proposed for their lower, average round and message requirements when compared to deterministic algorithms. In fact, the system no longer must be synchronous, and the number of rounds may be less than $t+1$. The idea is that at any particular round, there is a probability that the faulty PEs can thwart the consensus, but there is also a probability that Byzantine agreement will be reached. By randomizing the decision, it can be assured that different situations will constantly arise and that the faulty PEs will eventually fail

to break the consensus. Of course, the probability of this not happening in some expected amount of time must be considered when calculating the reliability of the system.

The first randomized Byzantine agreement algorithms were given separately by Rabin and Ben-Or for asynchronous systems [Rabin 1983; Ben-Or 1983].

Chor and Coan followed with an efficient algorithm for synchronous systems that takes an expected $O(t/\log n)$ rounds and $O(n^2 t/\log n)$ messages [Chor and Coan 1985]. The algorithm is completely distributed, and results in all the fault-free processors agreeing on a single binary value. Initially, each processor receives some input that it considers the correct value. This value is broadcast to all the other PEs. Upon receipt of these messages, each processor may change what it considers to be the correct value. If at least $n - t$ messages, where n is the number of processors and t is the maximum number of those which may be faulty, have the same value, then this value is considered correct, otherwise, the processor favors neither value as correct. A group of processors performs a random coin toss as described later. Once again, each processor broadcasts its favored value or an "undecided" message. A processor then counts the more popular value, other than "undecided," that it has received. Call this value v and let k be the count of messages with this value. If $k \geq n - t$ then the processor decides v is correct and exits the algorithm. If $n - t > k \geq t + 1$ then v becomes the favored value and the algorithm repeats itself. If $t + 1 > k$ then the processor assigns the value of the coin toss to its favored value, and the algorithm repeats.

The processors are divided into many disjoint groups of size g to perform the coin toss. At any round, one group will perform the toss. Each member tosses a coin and broadcasts the results. Thus, the toss of the group is the majority of the individual tosses. If more than half the group is faulty, then the toss will be to the advantage of the faulty PEs. Otherwise,

there is a sufficiently large probability that all the fault-free members of the group will produce the same toss overriding the faulty members. In any case, there are at most $2t/g$ disjoint groups with a majority of faulty PEs, so after at most that many tosses, there will be a toss whose result is to the disadvantage of the faulty processors with probability $1/2$ [Chor and Coan 1985; Shamir 1979].

Chor and Coan proved that a configuration of groups could be formed such that the coin tosses would be sufficiently random to foil any adversarial scheme. Moreover, they showed that at any particular round, there is a value of the random coin that will cause the algorithm to terminate, and that this value is unknown till the end of the round. Since the adversary model (see Section 3.1) does not allow the prediction of random variables, the algorithm will terminate with a probability arbitrarily close to one [Chor and Coan 1985].

Bracha proved that randomized algorithms for asynchronous systems require the number of PEs to be greater than or equal to $3t+1$ [Bracha 1987a]. For synchronous systems, though, an algorithm requiring $n \geq 2t + 1$ and only $O(\log n)$ rounds was given [Bracha 1987b].

Dispersed Joined Communications. A fundamental problem of fault-tolerant, multiprocessor systems is the acceptance of data from an external source. If the broadcast mechanism for this source were suffering from Byzantine faults, then each processor could receive a unique input value, and calculate a unique result. In this case, a fault-tolerant system would fail, despite the fact that none of its PEs had failed, because no correct result could be determined. This problem is known as the *Input Problem* [Krol 1991].

One solution to the Input Problem is to use Byzantine agreement after the set of inputs have arrived. This will make the system view of the input consistent. Krol introduced a set of algorithms called *Dispersed Joined Communication* (DJC) algorithms which solve the

Input Problem more efficiently, and in fact, which include the algorithm given by Pease *et al.* as a special case [Krol 1991; Pease *et al.* 1980].

Classes $\mathcal{A}(t, k, s, \mathbf{D}, \mathbf{N})$ of DJC algorithms are defined where t is the maximum number of faulty units, k is the number of rounds required, s is the source processor, $\mathbf{D}$ is the set of destination processors, and $\mathbf{N}$ is the set of all processors in a fully interconnected, synchronous system. The behavioral properties of the DJC algorithms are: 1) If the source and the destination are fault free, then the message received by the destination is equal, after deciphering, to the message held by the source; and 2) if two destination PEs hold different messages after the termination of the k-round DJC algorithm, then a message has traveled along a path of k distinct PEs all of which, including the source, behave maliciously. Krol proved that if $t \geq 1$, $k \geq 2$, $s \in \mathbf{N}$, $\mathbf{D} \subset \mathbf{N}$, and $|\mathbf{D}| \geq k + 1$ then the class of algorithms $\mathcal{A}(t, k, s, \mathbf{D}, \mathbf{N})$ is non-empty if and only if $|\mathbf{N}| \geq 2t + k$. He went on to give recursive procedures for designing these algorithms [Krol 1991].

Krol showed that the interactive consistency properties, which hold for all Byzantine agreement algorithms, hold for the class of algorithms $\mathcal{A}(t, k, s, \mathbf{D}, \mathbf{N})$ with $t \geq 1$, $k = t + 1$, and $\mathbf{D}=\mathbf{N}$. The flexibility of the DJC algorithms, though, can make them more efficient. Krol noted that the PEs performing Byzantine agreement send, to the other PEs, pieces of an error-correcting, coded message which are combined and deciphered by each fault-free PE. Namely, the code is a simple, repetition code that only requires a majority vote to decipher. That is, a processor will receive a series of values, say $1, 1, 1, 0$, and it will take a majority vote of these values to determine the actual value, in this case 1. The repetition code is not the most efficient error-correcting code, though, and the DJC algorithms take advantage of this. The DJC algorithms are configurable to either increase

coding, which reduces the number of messages but increases the minimal message size, or increase voting, i.e., decrease coding, which reduces the message size but increases the number of messages. Krol showed that for "practical" systems, i.e., $t \leq 3$, these DJC algorithms could be configured to outperform deterministic, synchronous, authenticated Byzantine agreement algorithms [Krol 1991].

5.2.2 System Requirements

Fischer *et al.* gave the very important result that distributed, deterministic (see the previous section on randomized algorithms) consensus was impossible in an *asynchronous* system with just one faulty processor [Fischer *et al.* 1985]. If no assumptions are made about the upper bound on how long a message may be in transit Δ, nor about the upper bound on the relative rates of processors Φ, then a single process running the consensus protocol could simply halt and delay the procedure indefinitely. In fact, Dolev *et al.* showed that if either Δ or Φ were unbounded, then consensus is impossible in the case of one fault [Dolev *et al.* 1987]. Dwork *et al.* explored the effects of *partial synchrony*, bounding Δ and Φ individually, on Byzantine agreement and gave algorithms that operate correctly on partially synchronous systems [Dwork *et al.* 1988]. Earlier work bounding Δ was done by Attiya [Attiya *et al.* 1984].

Asynchronous agreement. An interesting contrast to the need for synchronization was given by Dolev *et al.* who studied a protocol designed to reach a consensus on a real value rather than a binary value [Dolev *et al.* 1986]. The rules for agreement are that the final values obtained by each correct processor should be in the range of all the initial values, and that all values should be in agreement to within ε. The uses of such an algorithm include

clock synchronization and sensor stabilization. Using successive approximation techniques, terminating algorithms were given that would agree on arbitrarily close values in either synchronous or asynchronous environments.

Fekete also studied approximate agreement algorithms and gave asymptotically optimal procedures [Fekete 1991]. Moreover, these algorithms guarantee exact agreement after $t+1$ rounds and prior to that, try to locate faulty PEs to correct their outputs and cause a quicker convergence of the agreement.

Redundant broadcast channels. In many cases, the price of a Byzantine agreement protocol is too high. These protocols rely on redundancy in the time domain to mask faulty processors and slow down system performance. Babaoğlu and Drummond looked at masking these processors in the space domain [Babaoğlu and Drummond 1985]. Usually this is done by replicating the processors, which implies a voter and extra communication hardware, but it may also be done by putting redundancy into the network. In a broadcast system, such as an Ethernet or a Token Ring, a single bus is replaced by b busses and each processor is given ports to these lines. Cristian showed that in these systems, t faults may be masked with no more than $t+1$ messages on $b = t+1$ busses regardless of the total number of processors [Cristian 1989]. Moreover, some diagnostic information can be gathered from each broadcast.

6 DIAGNOSIS VERSUS AGREEMENT

System diagnosis and Byzantine agreement are two means to the same end. A population of faulty and fault-free processors must be reconciled to behave in a consistent, specified

manner. This can be done either by masking (Byzantine agreement) or by diagnosing (system diagnosis) the faulty processors. This section compares the two approaches to find where their costs lie.

Fault coverage is a fundamental measure of any fault-tolerant scheme. If no assumptions are made about failure characteristics, i.e., the system is Byzantine in nature, then the protocol must operate despite malicious attempts to stop it. Stronger failure semantics allow less robust algorithms, but require better-behaved environments.

Byzantine agreement protocols cope with weak (Byzantine faults) to strict (fail-stop, message authentication, private communication channels) failure requirements. On the other hand, system diagnosis is more concerned with the nature of the fault (permanent, transient or intermittent) than its consequences. This is because these solutions count on a test to reliably detect the fault, rather than on an approach which can ignore and tolerate the faulty PE. As a result, system diagnosis research has focused on maximizing the productivity of the test by looking at its nature (comparison versus directed) and the testing assignment.

Comparison test diagnostics and Byzantine agreement protocols run in parallel with the useful jobs of the system to achieve a high coverage at the cost of system performance. A diagnosis algorithm which uses the directed test approach must do so off-line, thus requiring recovery techniques to cope with errors. But this approach has an advantage in that there is no redundancy. Every processor which was considered working at the last test period may be given a unique job to perform. If it is later discovered to have failed, then the tasks dating back to the last successful test must be performed again. In terms of real-time systems, a high throughput is awarded for a long, worst-case delay in receiving a good

result. Both comparison test and Byzantine agreement algorithms require redundancy to detect and masks faults, respectively, but can meet stricter deadlines.

All consensus protocols must have access to an arbiter, but the implementation of this arbiter will strongly affect system performance. In centralized system diagnostic algorithms, an ultrareliable, unbiased processor collects test results, performs diagnostic calculations and returns processor conditions, i.e., faulty or fault free. As an arbiter, it must consistently and correctly determine who is faulty and who is not from a syndrome with possibly conflicting test results. Obviously, the reliability of these diagnosis schemes can be no greater than the reliability of the centralized observer and all the communication channels between it and the pool of processors. On the other hand, distributed system diagnosis and Byzantine agreement routines implement the arbiter within the protocol. In effect, every PE is its own arbiter. Now, reliability is no longer dependent on a single piece of hardware, but intuitively the algorithm will be less efficient in some manner. For example, there will no longer be a global snapshot of the condition of each processor, but rather a dynamic view from each PE which may be inconsistent with the view from the other processors.

It is difficult to compare the costs of the Byzantine agreement and system diagnosis protocols because of their differences and their unknowns. First, the size and nature of the task results affect the basic Byzantine agreement protocol as well as the comparison test in system diagnosis. Second, Byzantine agreement requires reliable communication between pairs of processors unless messages may be authenticated or the network is replicated (although, see Section 5.2.1). System diagnosis techniques may handle largely varying network topologies. Third, directed-testing, system diagnosis protocols are strongly affected by the frequency and complexity of the test. Byzantine agreement is dependent on the number

of agreements that need to be made. Fourth, researchers in the two areas have adopted different failure semantics for the faulty processor. This drastically affects efficiency. Fifth, centralized system diagnosis, processor membership and Byzantine agreement all maintain a consistent view of the system state. In other words, each fault-free PE is assured that its view of the system is the same as that of each other fault-free PE. Distributed system diagnosis can make the same assurance by adding timeouts or broadcasts. This is typically not examined, though, and would affect diagnosis efficiency.

Preparata *et al.* proved that for their model it is necessary that $n \geq 2t + 1$ for a system to be one-step t-diagnosable without repair [Preparata *et al.* 1967]. Depending on repairability, network structure, and the use of probabilistic approaches, though, this limit can change in either direction. Pease *et al.* proved that for Byzantine agreement, $n \geq 3t+1$ to mask t malicious processors [Pease *et al.* 1980; Lamport *et al.* 1982]. This limit is affected by assumptions on the authentication of messages, the synchrony of the system, network replication, and the use of probabilistic approaches. These limits must also be associated with a probability that consensus will actually fail when these worst case limits are reached. Babaoğlu showed that there is a nonzero probability of correct Byzantine agreement even when the number of faulty processors exceeds the resiliency bound [Babaoğlu 1987a; Babaoğlu 1987b]. Somani *et al.* had similar results for t-diagnosable systems [Somani *et al.* 1987]. A qualitative comparison of system diagnosis and Byzantine agreement is elusive.

In conclusion of this section, a graph of the various consensus models is presented. In Fig. 7, a node represents a model and an edge between two model nodes represents the major difference or extension between those models.

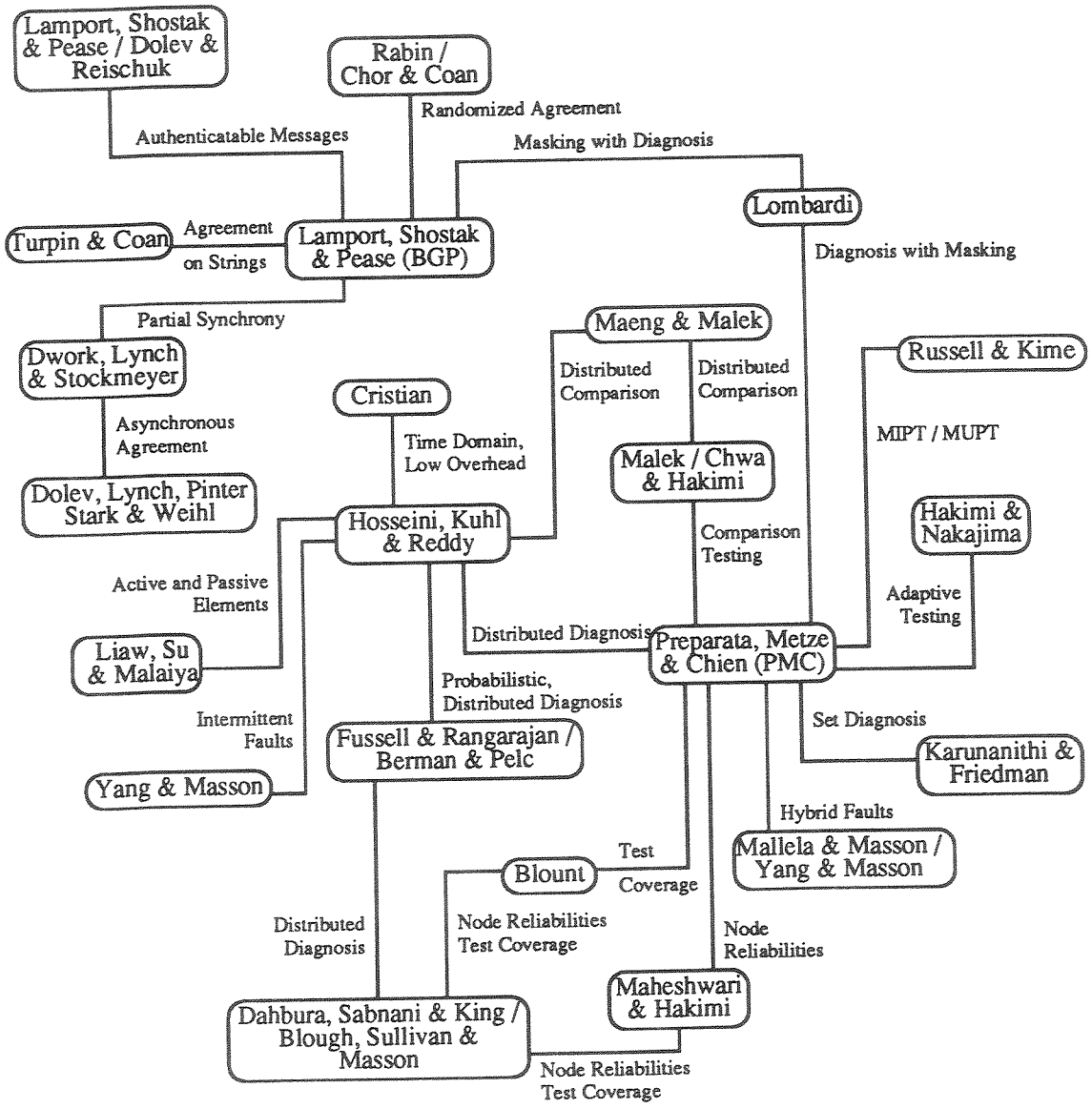


Figure 7: Consensus Models.

As Fig. 7 shows, very little work has been done that combines system diagnosis with Byzantine agreement despite the similarities of the two areas. Lombardi took the NMR technique for fault masking and cast it into the system diagnosis framework [Lombardi 1985]: n processors in a t-diagnosable system perform a number of tasks such that no task is done fewer than $t + 1$ times. Therefore, after the occurrence of any t faults, all tasks will have been performed correctly at least once. First, results are compared using NMR techniques to release a task as soon as possible, and to identify some faulty PEs. Next, if faulty PEs remain unidentified, Byzantine agreement is used to collect the results reported by each processor which are in turn used to diagnose the system. In other words, Byzantine agreement is used to distribute the centralized arbiter assumed in classic system diagnosis. A similar approach, i.e., disseminating the syndrome to avoid centralized analysis, was taken in the MuTeam approach discussed in Section 5.1.5 [Ciompi *et al.* 1981].

7 APPLYING CONSENSUS PROTOCOLS

The consensus problem takes many forms, and this section looks at some of the applications and implementations of this family of protocols. Preparata *et al.* and Pease *et al.* introduced the problem in fairly abstract terms. Their successors refined the models and assumptions to reflect more closely the conditions in a distributed computer network, thereby creating workable schemes for fault tolerance.

Often when ultrareliability is not an issue, *fail-stop processors* are assumed. That is, 1) a processor will halt rather than perform an unexpected state transition; 2) its status is apparent to other fail-stop processors; and 3) it utilizes stable storage. Schneider used a

consensus protocol to create a virtual fail-stop PE from a pool of less consistent processors [Schneider 1984]. Thus, when a highly available system is required, the algorithms designed for fail-stop processors can be used in conjunction with the virtual fail-stop processor protocol.

A protocol similar to Cristian's processor membership has been implemented at Tandem Computers to manage the processors within their systems [Cristian 1988; Tandem 1989]. The basic assumption is fail-stop processing. Every second, each processor p broadcasts an "I'm alive!" message. If processor p fails to receive a reply from processor q, it marks q as possibly faulty. At the next round of broadcasts, p also includes a message to itself. If q again fails to reply then p checks to see if it received its own message. If so, then q has failed and recovery procedures are initiated. Otherwise p halts and another processor will diagnose it as faulty after two more rounds of broadcasts.

Another example of a system using fail-stop processors is the Delta-4 Extra Performance Architecture [Barrett *et al.* 1990]. A process might have several copies running on distinct processors, but its results are issued from a single "leader." A failure of the leader will be uncovered through a timeout allowing a "follower" processor to use checkpointing or simultaneous execution to take the role as leader. This method is very similar to Cristian's and Tandem's membership protocols [Cristian 1988; Tandem 1989].

The Delta-4 systems also furnishes comparison testing of processes in the communication layer. Several processors executing the same process provide a single result to another group of processors. This is done using majority voting of signatures in the communication subsystem which allows fault diagnosis without the fail-stop requirement [Barrett *et al.* 1990].

Lamport uses consensus and strong failure semantics to create a database servicing multiple clients with the state machine approach [Lamport 1989]. The service a client requests must be committed or aborted in a manner which is apparent to the other clients. A consensus is formed on the state of the database. The design is based on fail-stop processors for a system of modest reliability. The algorithm is given in the guise of a ancient parliamentary system which happens to be very similar to a three-phase commit protocol.

Another popular application is the synchronization of a system of clocks. Synchronization is a consensus on an arbitrary time value at some precise, real time. Lamport and Melliar-Smith used Byzantine agreement solutions to synchronize clocks in a system with faulty processors and clocks [Lamport and Melliar-Smith 1984].

An early example of system diagnosis may be seen in the Micronet, a self-healing network for signal processing [DeGonia *et al.* 1978]. Multiple, homogeneous processors were connected by a bus. Spare processors were used as *standards* by which the others were checked, and *monitors* that actually compared results and reconfigured the system. A scheme was developed such that intermittently faulty PEs would be correctly diagnosed and taken off-line, and incorrectly diagnosed processors would be brought back on-line after retesting. Moreover, the authors recognized that faulty PEs might still be able to perform certain functions and so diagnosed them according to functionality loss. The entire system was a hierarchical composition of these diagnosable, bus-based subsystems. Testing was performed within and between levels such that eventually the entire system and all of its functions would be tested.

Agrawal suggested a broadcast network connecting a pool of processors and controlled

by a reliable scheduler and diagnostics manager as a fault-tolerant architecture [Agrawal 1985]. Diagnosis and fault tolerance is achieved by way of comparison testing. The scheduler assigns tasks to different processors and the diagnostics manager compares these results. If a match occurs, the result is considered correct, otherwise the task is rescheduled until a match is found. In other words, the number of processors involved in the consensus is increased from two until any pair of them match results.

Philips has incorporated Byzantine agreement into their fault-tolerant switching system via the DJC algorithms given by Krol (see Section 5.2.1) [Krol 1991]. In this $(4,2)$-concept fault-tolerant computer, there are four redundant processors accessing a single, error-correcting coded memory divided into four, separate modules. Krol used the DJC algorithms to ensure that the four processors would receive the same inputs whether the system was connected to a single source or to another fault-tolerant multiprocessor [Krol 1991].

At the University of Erlangen-Nürnberg, system diagnosis was used for the DIRMU (DIstributed Reconfigurable MUltiprocessor) system which contained 25 PEs [Maehle *et al.* 1986]. The algorithm implemented was similar to **SELF3** given in [Hosseini *et al.* 1984] and was able to diagnose PEs and communication links as well as determine the intact configuration of the multiprocessor. Maehle reported that system diagnosis worked without problem over the five year (1985-1990) lifespan of the DIRMU system [Maehle 1991].

At Carnegie-Mellon University, system diagnosis was applied to an Ethernet connecting over 100 workstations by adopting the distributed diagnosis algorithm **NEW_SELF** given by Hosseini *et al.* [Bianchini *et al.* 1990; Hosseini *et al.* 1984]. The CMU diagnosis solution uses time to synchronize tests and detect faulty PEs; it allows for PEs to join and leave the

set of fault-free PEs; it calls for testing reconfiguration after PE failures; and it assumes system stability to minimize overhead. It does not test for faulty communication links.

The implementation uses self-tests initiated by outside PEs. When a processor A tests another processor B, it actually sends a request to B that it test itself. B spawns a subprocess which reads from the disk and executes floating point operations. Thus, the operation of the disk, the operating system software, and some portion of the CPU are tested without starving any productive tasks of resources. The diagnosis algorithm treats the result of this test as complete and the authors reported that this simple test caught every processor fault that occurred over a period of two years.

One of the major concerns of this work was the communication overhead required by the **NEW_SELF** algorithm. A significant reduction may be had by combining test requests and results. If processor p is being tested by all of its neighbors, then it need run the self-test only once and report the same result to these testers. If processor p is testing some processor q and receives a request for another test of q, then p may combine these requests and disseminate the test result accordingly. These simplifications work for two reasons: the benign failure of processors, and the implementation of the test as a self-test.

Another technique used for reducing message transmissions was to take advantage of the stability of the system. That is, a stable system has no faults or joins occurring, and thus, requires no new diagnosis. The authors devised an event-driven algorithm in which diagnostic information is not passed unless a fault or join occurs.

Table 2 compares major characteristics of **NEW_SELF** [Hosseini *et al.* 1984], Cristian's processor membership [Cristian 1991b], and the CMU diagnosis solution [Bianchini *et al.* 1990].

Table 2: A Comparison of Diagnosis Strategies.

	NEW_SELF	Processor Membership	CMU Diagnosis
System Model	Graph model of physical interconnection network with test graph embedded on it.	Physical model with communication delays and completely connected logical graph with test graph embedded on it.	Graph model of physical interconnection network and completely connected logical graph with test graph embedded on it.
Physical Network	Arbitrary point to point network.	Arbitrary point to point network.	Viewed as a bus based network.
The Test	Fault-tolerant mechanisms included with the PE (self-testing circuits, monitors, watchdogs) report PE status on demand.	Self-tests implied, on top of these is the ability to send and receive messages via a reliable broadcast in a timely manner.	Self- test process tests OS process handling, disk I/O, and floating point unit operation. Must be able to send and receive test results via reliable communications in a timely manner.
Tested By	Physically adjacent neighbors.	Logical neighbors. Preferably physical neighbors.	Closest neighbors on one side of bus, also logical neighbors.
Diagnosis Time	O(diameter of testing graph)	O(diameter of physical graph) or O(number of PEs)	O(diameter of testing graph)
Test Rounds	Asynchronous, but finite in length.	Synchronized by a global clock which implies timing fault coverage.	Synchronous.

Table 2: A Comparison of Diagnosis Strategies *(continued)*.

	NEW_SELF	Processor Membership	CMU Diagnosis
Fault Classes	Faulty processing elements and communication links.	Crash faults, omission faults, timing faults, Byzantine faults (through atomic broadcast).	Faulty PE, omission faults, timing faults.
Fault Validation	Test the PE reporting the fault at the next test round.	All PEs broadcast their presence to reform the group.	Immediately test the PE reporting the fault.
Joining	New or repaired PEs clear their old messages. Diagnosability is affected.	A new or repaired PE initiates a group broadcast to reform the membership and get the membership list.	Test graph is reconfigured. The new PE gets the system diagnosis from the PEs it tests.
Leaving	PE is diagnosed as faulty and is ignored by the fault-free PEs. System diagnosability is decreased.	A faulty PE is detected and a group broadcast occurs so the membership reconfigures. Diagnosability based on new membership.	A faulty PE may be removed from the system causing a reconfiguration of the test graph and possibly the same diagnosability.
System Stability	Redundant diagnosis messages are sent with high system overhead even without faults.	Neighborhood Surveillance Protocol only has test communications between neighbors when there are no faults or joins.	Send only test requests and results when there are no faults or joins. If there is a fault or join then pass on the new diagnosis information.

Bianchini and Buskens continued the experiment at CMU with an adaptive diagnosis algorithm again based on the **NEW_SELF** algorithm [Bianchini and Buskens 1991; Hosseini *et al.* 1984]. Previously, the testing assignment for the system was defined a priori, but in this case, tests are performed as indicated by the current fault set. The result is correct diagnosis despite an unlimited number of faulty PEs, as well as further reduced communication overhead.

8 FUTURE RESEARCH

The trend in consensus problem research has been towards incorporating higher levels of realism into the solutions with the ultimate goal being implementation. Probabilistic approaches have emerged due to their efficiency compared with deterministic solutions. Fault models and test models that aim to describe actual events in a distributed computing environment have also been examined. This section outlines many points of practicality for creators of future consensus protocols.

Losing and gaining processor elements. An admission that processors will fail is made simply by the pursuit of consensus algorithms. When a PE fails, it should be removed from the fault-free membership to improve diagnosability and diagnosis efficiency. In large distributed systems, there will be new or repaired fault-free computers to be added to the system in an on-line manner. Therefore, the ability to add elements without disrupting diagnosis is required, even if it is simply to accommodate users who turn their workstations off in the evening and on in the morning. This point is explicitly considered by Cristian and Hosseini *et al.* [Cristian 1988; Hosseini *et al.* 1984]. Little work has been done to

incorporate the diagnosis of failed PEs into Byzantine agreement algorithms.

Imperfect tests. One of the most disturbing phrases in the system diagnosis research is "Processor A tests Processor B." Testing can be done in simple ways by using comparison testing [Malek 1980; Chwa and Hakimi 1981], fault detection mechanisms like watchdog timers as in the Tandem '16 computers, or simple diagnostic processes which run on the target processor [Bianchini *et al.* 1990]. The CMU experiment has shown that a simple self-test gives practical fault coverage. This coverage may not be enough for certain systems, though, leaving open the question of whether direct-testing schemes are sufficient in ultrareliable systems.

Distributed diagnosis. Fault tolerance is the ultimate goal of system diagnosis; therefore, centralizing the diagnosis function, and thereby creating a "weak link," is usually unacceptable. Diagnosis should be performed by each processing element to increase fault tolerance and diagnostic responsiveness.

General network topologies. Related to distributed diagnosis is the structure of the network. Much work has been done searching for classes of testing graphs that satisfy the requirements of certain diagnosis algorithms, but more needs to be done to learn how to overlay these testing assignments efficiently on physical communication systems [Fussell and Rangarajan 1989]. This includes adapting to changing topologies due to data link failures or repairs.

System overhead. As always, algorithm efficiency is of interest. It is necessary to reduce the number and size of the messages until their effect on other communications is virtually transparent. Otherwise, diagnosis itself might become the system bottleneck. Currently, Byzantine agreement protocols can handle only a few faults, i.e., three or fewer, before the

overhead becomes unbearable [Krol 1991].

Very Large Systems. The desire to connect equipment is overwhelming and the result has been very large computer networks such as Arpanet and Internet. Obviously, no single PE needs to know the status of every other element in the system. Therefore, one should consider hierarchical schemes of diagnosis and computing in which a processor knows if a server pool is still operating. Information would be on the basis of what partition of servers is desired, while in that partition, diagnosis or masking would occur in one of the many manners described in this paper. That is, each group of servers would have its own consensus protocol dependent on the goal of that partition. At the next level, partitions would diagnose each other to determine if the number of faulty elements within a group had precluded correct diagnosis or masking.

Time. Analysis in the time domain is perhaps the most important characteristic of the work done in processor membership [Cristian 1991b]. It is reasonable, with the increasing popularity of responsive or real-time, fault-tolerant systems, to construct diagnosis algorithms in the time domain. Many processor failures are detectable with timing constraints. A crash or omission fault will cause a receiver to timeout, and a timing fault implies a receiver received a message when it was not expecting one. In all cases, the test of a processor should cover the ability to send and receive messages in a timely manner. Fischer *et al.* showed the importance of synchronization and time, and that ignorance of the time performance of a processor can render Byzantine agreement impossible [Fischer *et al.* 1985].

Bounded faults. The notion of t-diagnosability is rather conservative in many realms. In a small network of common workstations, which tend to be highly reliable, more than two or three faults could be unreasonable. Given such constraints, very efficient diagnosis

algorithms might exist. More general algorithms could still be required for low yield, wafer scale systems or non-repairable, "mission" systems. One solution is to develop algorithms which make no assumptions about the number of faulty PEs, but which are almost always correct such as [Dahbura *et al.* 1987].

Network characterization. Little has been reported on the frequency and types of faults experienced in large, distributed networks. Practical information of this sort could greatly increase the potential usefulness of the consensus protocols that have been surveyed. For example, the class of Byzantine faults is far-flung and includes situations that are arguably nonexistent. The rarity of a faulty PE sending two different, yet valid, messages to two of its neighbors, when the communications should have been identical, needs to be quantified.

9 CONCLUSIONS

In many cases, a consensus protocol dictates certain characteristics of its target system. It may assume private communications or a centralized arbiter among others. Moreover, the protocol can influence the type of decision to be made. An algorithm may operate on the basis that a single result is incorrect or that a processor is untrustworthy. In the future, consensus protocols will impact decision systems and decision making rather than vice versa. Their presence at all levels of computing guarantees a high priority of efficiency. A natural dilemma arises: either researchers examine a generic framework which will apply to any system, e.g., computing, economics, government, etc., which is reminiscent of Byzantine agreement, or researchers assume a computing environment, develop requirements for high performance, and then find an analogy for, or start anew on other systems, which is the

direction system diagnosis research is taking. Whatever the outcome, these fundamental consensus algorithms will persist despite the underlying framework and will affect systems of multiple computing elements to come. Thus, the work presented in this paper is an outline for a new method of fault-tolerant system design.

10 SUMMARY

The importance of the consensus problem stems from its ubiquitous nature in distributed fault-tolerant computing. It is alternately veiled as a synchronization problem, a reliable communication protocol, a resource allocator, a task scheduler or a diagnosis/reconfiguration scheme, among others. Playing a role in so many aspects of computing makes it fundamental. In this paper, the particular consensus application of producing a correct result in an environment that includes faulty processors was examined. Two schools of thought reign: *system diagnosis* in which a population keeps tabs on its faulty processors, and *Byzantine agreement* in which faulty processors are masked by an abundance of fault-free constituents. The history of these two areas was outlined with the hopes that future researchers would reconcile both fields or at least draw from the more appropriate source depending on the application. The paper also discussed how these ideas are being put to work in real systems. Finally, directions for more work were proposed with the belief that practicality and implementability will be of high priority.

11 ACKNOWLEDGMENTS

Miroslaw Malek was supported during this work by ONR Grant N00014-88-K-0543 and NASA Grant NAG9-426. Michael Barborak was supported during this work by a Microelectronics and Computer Development Fellowship administered through the University of Texas at Austin.

REFERENCES

[Agrawal 1985] P. Agrawal, "RAFT: A Recursive Algorithm for Fault Tolerance," *International Conference on Parallel Programming*, pp. 814-821, 1985.

[Attiya *et al.* 1984] C. Attiya, D. Dolev, J. Gil, "Asynchronous Byzantine Consensus," *Third Annual ACM Symposium on Principles of Distributed Computing*, pp. 119-133, 1984.

[Babaoğlu and Drummond 1985] Ö. Babaoğlu, R. Drummond, "Streets of Byzantium: Network Architectures for Fast Reliable Broadcasts," *IEEE Transactions on Software Engineering*, Vol. SE-11, No. 6, pp. 546-554, June 1985.

[Babaoğlu 1987a] Ö. Babaoğlu, "Stopping Times of Distributed Consensus Protocols: A Probabilistic Approach," *Information Processing Letters*, Vol. 25, No. 3, pp. 163-169, May 1987.

[Babaoğlu 1987b] Ö. Babaoğlu, "On the Reliability of Consensus-Based Fault-Tolerant Distributed Computing Systems," *ACM Transactions on Computer Systems*, Vol. 5, No. 3, pp. 394-416, November 1987.

[Bagchi and Hakimi 1991] A. Bagchi, S. Hakimi, "An Optimal Algorithm For Distributed System Level Diagnosis", *Twenty-First International Symposium on Fault-Tolerant Computing*, pp. 214-221, 1991.

[Bar-Noy and Dolev 1991] A. Bar-Noy, D. Dolev, "Consensus Algorithms with One-Bit Messages," *Distributed Computing*, Vol. 4, pp. 105-110, 1991.

[Barrett *et al.* 1990] P. Barrett, A. Hilborne, P. Bond, D. Seaton, P. Verissimo, L. Rodrigues, N. Speirs, "The Delta-4 Extra Performance Architecture (XPA)," *Twentieth International Symposium on Fault-Tolerant Computing*, pp. 481-488, 1990.

[Barsi *et al.* 1976] F. Barsi, F. Grandoni, P. Maestrini, "A Theory of Diagnosability of Digital Systems," *IEEE Transactions on Computers*, Vol. C-25, No. 6, pp. 585-593, June 1976.

[Ben-Or 1983] M. Ben-Or, "Another Advantage of Free Choice: Completely Asynchronous Agreement Protocols," *Second Annual ACM Symposium on Principles of Distributed Computing*, pp. 27-30, 1983.

[Berman and Pelc 1990] P. Berman, A. Pelc, "Distributed Probabilistic Fault Diagnosis for Multiprocessor Systems," *Twentieth International Symposium on Fault-Tolerant Computing*, pp. 340-346, 1990.

[Bertsekas and Gallager 1987] D. Bertsekas, R. Gallager, *Data Networks*, pp. 340-355, Prentice-Hall, Inc., New Jersey, 1987.

[Bianchini *et al.* 1990] R. Bianchini, K. Goodwin, D. Nydick, "Practical Application and Implementation of Distributed System-Level Diagnosis Theory," *Twentieth International Symposium on Fault-Tolerant Computing*, pp. 332-339, 1990.

[Bianchini and Buskens 1991] R. Bianchini, R. Buskens, "An Adaptive Distributed System-Level Diagnosis Algorithm and Its Implementation, " *Twenty-First International Symposium on Fault-Tolerant Computing*, pp. 222-229, 1991.

[Birman and Joseph 1987] K. Birman, T. Joseph, "Reliable Communication in the Presence of Faults," *ACM Transactions on Computer Systems*, Vol. 5, No. 1, pp. 47-76, February 1987.

[Blecher 1983] P. Blecher, "On a Logical Problem," *Discrete Mathematics*, Vol. 43, pp. 107-110, 1983.

[Blough *et al.* 1988] D. Blough, G. Sullivan, G. Masson, "Almost Certain Diagnosis for Intermittently Faulty Systems," *Eighteenth International Symposium on Fault-Tolerant Computing*, pp. 260-265, June 1988.

[Blough *et al.* 1989] D. Blough, G. Sullivan, G. Masson, "Fault Diagnosis for Sparsely Interconnected Multiprocessor Systems," *Nineteenth International Symposium on Fault-Tolerant Computing*, pp. 62-69, June 1989.

[Blough and Pelc 1990] D. Blough, A. Pelc, "Reliable Diagnosis and Repair in Constant-Degree Multiprocessor Systems," *Twentieth International Symposium on Fault-Tolerant Computing*, pp. 316-323, 1990.

[Blount 1977] M. Blount, "Probabilistic Treatment of Diagnosis in Digital Systems," *Seventh International Symposium on Fault-Tolerant Computing*, pp. 72-77, June 1977.

[Bracha 1987a] G. Bracha, "Asynchronous Byzantine Agreement Protocols," *Inform. and Comp.*, Vol. 75, pp. 130-143, November 1987.

[Bracha 1987b] G. Bracha, "An $O(\log n)$ Expected Rounds Randomized Byzantine Generals Protocol," *Journal of the ACM*, Vol. 34, pp. 910-920, 1987.

[Chor and Coan 1985] B. Chor, B. Coan, "A Simple and Efficient Randomized Byzantine Agreement Algorithm," *IEEE Transactions on Software Engineering*, Vol. SE-11, No. 6, pp. 531-539, June 1985.

[Chwa and Hakimi 1981a] K. Chwa, S. Hakimi, "Schemes for Fault-Tolerant Computing: A Comparison of Modularly Redundant and t-diagnosable Systems," *Information and Control*, Vol. 49, pp. 212-238, 1981.

[Chwa and Hakimi 1981b] K. Chwa, S. Hakimi, "On Fault Identification in Diagnosable Systems," *IEEE Transactions on Computers*, Vol. C-30, No. 6, pp. 414-422, June 1981.

[Ciompi 1981] Ciompi, F. Grandoni, L. Simoncini, "Distributed Diagnosis in Multiprocessor Systems: The MuTeam Approach," *Eleventh International Symposium on Fault-Tolerant Computing*, pp. 25-29, June 1981.

[Coan 1988] B. Coan, "Efficient Agreement Using Fault Diagnosis," *Proceedings of the* 26^{th} *Allerton Conference on Communication, Control and Computing*, pp. 663-672, 1988.

[Cristian 1989] F. Cristian, "Synchronous Atomic Broadcast for Redundant Broadcast Channels," *IBM Research Report* RJ 7203, December 1989.

[Cristian 1990] F. Cristian, "Fault-Tolerance in the Advanced Automation System," *IBM Research Report* RJ 7424 (69595), April 16, 1990.

[Cristian 1991a] F. Cristian, "Understanding Fault-Tolerant Distributed Systems," *Communications of the ACM*, Vol. 34, No. 2, Feb. 1991.

[Cristian 1991b] F. Cristian, "Reaching Agreement on Processor-Group Membership in Synchronous Distributed Systems," *Distributed Computing* Vol. 4, pp. 175-187, 1991.

[Cristian *et al.* 1986] F. Cristian, H. Aghili, R. Strong, D. Dolev, "Atomic Broadcast: From Simple Message Diffusion to Byzantine Agreement," *IBM Technical Report* RJ 5244 (54244), July 30, 1986 (Also *Fifteenth International Symposium on Fault-Tolerant Computing*, pp. 200-206, 1985).

[Dahbura 1986] A. Dahbura, "An Efficient Algorithm for Identifying the Most Likely Fault Set in a Probabilistically Diagnosable System," *IEEE Transactions on Computers*, Vol. C-35, No. 4, pp. 354-356, April 1986.

[Dahbura 1988] A. Dahbura, "System-Level Diagnosis: A Perspective for the Third Decade," *AT&T Bell Laboratories Report. Concurrent Computations: Algorithms, Architecture, and Technology*, (S. Tewksbury, B. Dickinson, S. Schwartz, eds.), Plenum Press, 1988.

[Dahbura and Masson 1983a] A. Dahbura, G. Masson, "Greedy Diagnosis of Hybrid Fault Situations," *IEEE Transactions on Computers*, Vol. C-32, No. 8, pp. 777-782, August 1983.

[Dahbura and Masson 1983b] A. Dahbura, G. Masson, "Greedy Diagnosis of an Intermittent-Fault/Transient-Upset Tolerant System Design," *IEEE Transactions on Computers*, Vol. C-32, No. 10, pp. 953-957, October 1983.

[Dahbura and Masson 1984a] A. Dahbura, G. Masson, "An $O(n^{2.5})$ Fault Identification Algorithm for Diagnosable Systems," *IEEE Transactions on Computers*, Vol. C-33, No.

6, pp. 486-492, June 1984.

[Dahbura and Masson 1984b] A. Dahbura, G. Masson, "A Practical Variation of the $O(n^{2.5})$ Fault Diagnosis Algorithm," *Fourteenth International Symposium on Fault-Tolerant Computing*, pp. 428-433, June 1984.

[Dahbura *et al.* 1985a] A. Dahbura, J. Laferrera, L. King, "A Performance Study of System-Level Fault Diagnosis Algorithms (I)," *Fourth International Conference on Computers and Communications*, pp. 469-473, March 1985.

[Dahbura *et al.* 1985b] A. Dahbura, G. Masson, C. Yang, "Self-Implicating Structures for Diagnosable Systems," *IEEE Transactions on Computers*, Vol. C-34, No. 8, pp. 718-723, August 1985.

[Dahbura *et al.* 1987] A. Dahbura, K. Sabnani, L. King, "The Comparison Approach to Multiprocessor Fault Diagnosis," *IEEE Transactions on Computers*, Vol. C-36, No. 3, pp. 373-378, March 1987.

[Dahbura *et al.* 1989] A. Dahbura, K. Sabnani, W. Hery, "Spare Capacity as a Means of Fault Detection and Diagnosis in Multiprocessor Systems," *IEEE Transactions on Computers*, Vol. C-38, No. 6, June 1989, pp. 881-891.

[DeGonia *et al.* 1978] P. DeGonia, R. Witt, D. Lampe, E. Cole, "Micronet - A Self-Healing Network for Signal Processing," *Govt. Microcircuit Appl. Conf.*, pp. 370-375, Nov. 1978.

[Deo 1974] N. Deo, *Graph Theory with Applications to Engineering and Computer Science*, Prentice-Hall, Englwood Cliffs, NJ, 1974.

[Dolev 1981] D. Dolev, "Unanimity in an Unknown and Unreliable Environment," *Twenty-Second Symposium on the Foundations of Computer Science*, pp. 159-168, 1981.

[Dolev 1982] D. Dolev, "The Byzantine Generals Strike Again," *Journal of Algorithms* , Vol. 3, pp. 14-30, 1982.

[Dolev and Reischuk 1985] D. Dolev, R. Reischuk, "Bounds on Information Exchange for Byzantine Agreement," *Journal of the ACM*, Vol. 32, No. 1, pp. 191-204, January 1985.

[Dolev *et al.* 1986] D. Dolev, N. Lynch, S. Pinter, E. Stark, W. Weihl, "Reaching Approximate Agreement in the Presence of Faults," *Journal of the ACM*, Vol. 33, No. 3, pp. 499-516, July 1986.

[Dolev *et al.* 1987] D. Dolev, C. Dwork, L. Stockmeyer, "On the Minimal Synchronism Needed for Distributed Consensus," *Journal of the ACM*, Vol. 34, No. 1, pp. 77-97, January 1987.

[Dwork *et al.* 1988] C. Dwork, N. Lynch, L. Stockmeyer, "Consensus in the Presence of Partial Synchrony," *Journal of the ACM*, Vol. 35, No. 2, pp. 288-323, April 1988.

[Fekete 1991] A. Fekete, "Asymptotically Optimal Algorithms for Approximate Agreement," *Distributed Computing*, Vol. 4, pp. 9-29, 1991.

[Fischer and Lynch 1982] M. Fischer, N. Lynch, "A Lower Bound for the Time to Assure Interactive Consistency," *Information Processing Letters*, Vol. 14, No. 4, pp. 183-186, June 1982.

[Fischer *et al.* 1985] M. Fischer, N. Lynch, M. Paterson, "Impossibility of Distributed Consensus with One Faulty Process," *Journal of the ACM*, Vol. 32, No. 2, pp. 374-382, April 1985.

[Friedman 1975] A. Friedman, "A New Measure of Digital System Diagnosis," *Fifth International Conference on Fault-Tolerant Computing*, pp. 167-169, 1975.

[Friedman and Simoncini 1980] A. Friedman, L. Simoncini, "System-Level Fault Diagnosis," *Computer*, pp. 47-53, March 1980.

[Fujiwara and Kinoshita 1978] H. Fujiwara, K. Kinoshita, "On the Computational Complexity of System Diagnosis," *IEEE Transactions on Computers*, Vol. C-27, No. 10, pp. 881-885, October 1978.

[Fussell and Rangarajan 1989] D. Fussell, S. Rangarajan, "Probabilistic Diagnosis of Multiprocessor Systems with Arbitrary Connectivity," *Nineteenth International Symposium on Fault-Tolerant Computing*, pp. 560-565, June 1989.

[Hakimi and Amin 1974] S. Hakimi, A. Amin, "Characterization of Connection Assignment of Diagnosable Systems," *IEEE Transactions on Computers*, Vol. C-23, No. 1, pp. 86-88, January 1974.

[Hakimi and Nakajima 1984] S. Hakimi, K. Nakajima, "On Adaptive System Diagnosis," *IEEE Transactions on Computers*, Vol. C-33, No. 3, pp. 234-240, March 1984.

[Harary *et al.* 1988] F. Harary, J. Hayes, H. Wu, "A Survey of the Theory of Hypercube Graphs," *Comput. Math. Applic.*, Vol. 15, No. 4, pp. 277-289, 1988.

[Holt and Smith 1981] C. Holt, J. Smith, "Diagnosis of Systems with Asymmetric Invalidation," *IEEE Transactions on Computers*, Vol. C-30, No. 9, pp. 679-690, September 1981.

[Hosseini *et al.* 1984] S. Hosseini, J. Kuhl, S. Reddy, "A Diagnosis Algorithm for Distributed Computing Systems with Dynamic Failure and Repair," *IEEE Transactions on Computers*, Vol. C-33, No. 3, pp. 223-233, March 1984.

[Hosseini *et al.* 1985] S. Hosseini, J. Kuhl, S. Reddy, "On Self Fault-Diagnosis of the Distributed Systems," *Fifteenth International Symposium on Fault-Tolerant Computing*, pp. 30-35, June 1985.

[Huang *et al.* 1989] S. Huang, J. Xu, T. Chen, "Characterization and Design of Sequentially t-Diagnosable Systems," *Nineteenth International Symposium on Fault-Tolerant Computing*, pp. 554-559, June 1989.

[Johnson and Malek 1989] A. Johnson, M. Malek, "Fault Classification: The First Step in Fault-Tolerant Design," *IBM Technical Report* TR 51.0493, January 1989.

[Kameda *et al.* 1975] T. Kameda, S. Toida, F. Allan, "A Diagnosing Algorithm for Networks," *Information and Control*, Vol. 29, pp. 141-148, 1975.

[Karunanithi and Friedman 1977] S. Karunanithi, A. Friedman, "System Diagnosis with t/s Diagnosability," *Seventh International Symposium on Fault-Tolerant Computing*, pp. 65-71, June 1977.

[Kavianpour and Friedman 1978] A. Kavianpour, A. Friedman, "Efficient Design of Easily Diagnosable Systems," *Proceedings of the Third USA-Japan Computer Conference*,

pp. 251-257, 1978.

[Kim and Yang 1986] K. Kim, S. Yang, "Fault Tolerance Mechanisms in Real-Time Distributed Operating Systems: An Overview," *Pacific Computer Communications '85*, (K. Kim, K. Chon, C. Ramamoorthy, eds.), Elsevier Science Publishers, pp. 239-248, 1986.

[Kime 1970] C. Kime, "An Analysis Model for Digital System Diagnosis," *IEEE Transactions on Computers*, Vol. C-19, No. 11, pp. 1063-1073, November 1970.

[Kime 1986] C. Kime, "System Diagnosis," *Fault Tolerant Computing: Theory and Techniques*, (D. Pradhan, ed.), Prentice-Hall, 1986.

[Kohda and Abiru 1988] T. Kohda, K. Abiru, "A Recursive Procedure for Optimally Designing a Hybrid Fault Diagnosable System," *Eighteenth International Symposium on Fault-Tolerant Computing*, pp. 272-277, June 1988.

[Kreutzer and Hakimi 1983] S. Kreutzer, S. Hakimi, "Adaptive Fault Identification in Two New Diagnostic Models," *Proceedings of the* 21^{st} *Allerton Conference on Communication, Control and Computing*, pp. 353-362, 1983.

[Kreutzer and Hakimi 1987] S. Kreutzer, S. Hakimi, "System-Level Fault Diagnosis: A Survey," *Microprocessing and Microprogramming*, No. 20, pp. 323-330, 1987.

[Kreutzer and Hakimi 1988] S. Kreutzer, S. Hakimi, "Distributed Diagnosis and the System User," *IEEE Transactions on Computers*, Vol. 37, No. 1, pp. 71-78, January 1988.

[Krol 1991] T. Krol, "A Generalization of Fault-Tolerance Based on Masking," Ph.D. Thesis, Eindhoven University of Technology, September 1991.

[Kuhl and Reddy 1980] J. Kuhl, S. Reddy, "Distributed Fault-Tolerance for Large Multiprocessor Systems," *Proc. Seventh Annual Symposium on Computer Architecture*, pp. 23-30, 1980.

[Kuhl and Reddy 1981] J. Kuhl, S. Reddy, "Fault-Diagnosis in Fully Distributed Systems," *Eleventh International Symposium on Fault-Tolerant Computing*, pp. 100-105, June 1981.

[Lamport *et al.* 1982] L. Lamport, R. Shostak, M. Pease, "The Byzantine Generals Problem," *ACM Transactions on Programming Languages and Systems*, Vol. 4, No. 3, pp. 382-401, July, 1982.

[Lamport and Melliar-Smith 1984] L. Lamport, P. Melliar-Smith, "Byzantine Clock Synchronization," *Third ACM Symposium on Principles of Distributed Computing*, pp. 68-74, 1984.

[Lamport 1989] L. Lamport, "The Part-Time Parliament," *Digital Systems Research Center Technical Report*, No. 49, September 1989.

[Laranjeira *et al.* 1991] L. Laranjeira, M. Malek, R. Jenevein, "On Tolerating Faults in Naturally Redundant Algorithms," *Tenth Symposium on Reliable Distributed Systems*, Pisa, Italy, pp. 118-127, September 1991.

[Lee and Shin 1990] S. Lee, K. Shin, "Optimal Multiple Syndrome Probabilistic Diagnosis," *Twentieth International Symposium on Fault-Tolerant Computing*, pp. 324-331, 1990.

[Liaw *et al.* 1982] C. Liaw, S. Su, Y. Malaiya, "Self-Diagnosis of Non-Homogeneous Distributed Systems," *Twelfth International Symposium on Fault-Tolerant Computing*,

pp. 349-352, June 1982.

[Lombardi 1985] F. Lombardi, "Diagnosable Systems for Fault Tolerant Computing," *Fifteenth International Symposium on Fault-Tolerant Computing*, pp. 42-47, June 1985.

[Lynch *et al.* 1982] N. Lynch, M. Fischer, R. Fowler, "A Simple and Efficient Byzantine Generals Algorithm," 2^{nd} *Annual Symposium on Reliability in Distributed Software and Database Systems*, pp. 46-52, July 1982.

[Maehle 1991] E. Maehle, Private communication, October 1991.

[Maehle *et al.* 1986] E. Maehle, K. Moritzen, K. Wirl, "A Graph Model for Diagnosis and Reconfiguration and Its Application to a Fault-Tolerant Multiprocessor System," *Sixteenth International Symposium on Fault-Tolerant Computing*, Vienna, Austria, pp. 292-297, 1986.

[Maeng and Malek 1981] J. Maeng, M. Malek, "A Comparison Connection Assignment for Self-Diagnosis of Multiprocessor Systems," *Eleventh International Symposium on Fault-Tolerant Computing*, pp. 173-175, June 1981.

[Maheshwari and Hakimi 1976] S. Maheshwari, S. Hakimi, "On Models for Diagnosable Systems and Probabilistic Fault Diagnosis," *IEEE Transactions on Computers*, Vol. C-25, No. 3, pp. 228-236, March 1976.

[Malek 1980] M. Malek, "A Comparison Connection Assignment for Diagnosis of Multiprocessor Systems," *Proc. Seventh Annual Symposium on Computer Architecture*, pp. 31-36, May 1980.

[Malek 1991] M. Malek, "Responsive Systems: A Marriage between Real Time and Fault Tolerance," *Fifth International Symposium on Fault-Tolerant Computing Systems*, Nurenberg, 1991.

[Malek and Liu 1980] M. Malek, K. Liu, "Graph Theory Models in Fault Diagnosis and Fault Tolerance," *Design Automation & Fault-Tolerant Computing*, Volume III, Issue 3/4, Computer Science Press, pp. 155-169, 1980.

[Mallela and Masson 1978] S. Mallela, G. Masson, "Diagnosable Systems for Intermittent Faults," *IEEE Transactions on Computers*, Vol. C-27, No. 6, pp. 560-566, June 1978.

[Mallela 1980] S. Mallela, "On Diagnosable Systems with Simple Algorithms," *Proceedings 1980 Conference on Information Science and Systems*, Princeton University, pp. 545-549, March 1980.

[Mallela and Masson 1980] S. Mallela, G. Masson, "Diagnosis Without Repair for Hybrid Fault Situations," *IEEE Transactions on Computers*, Vol. C-29, No. 6, pp. 461-470, June 1980.

[Maxemchuk and Dahbura 1986] N. Maxemchuk, A. Dahbura, "Optimal Diagnosable System Design Using Full-Difference Triangles," *IEEE Transactions on Computers*, Vol. C-35, No. 9, pp. 837-839, September 1986.

[Meyer and Masson 1978] G. Meyer, G. Masson, "An Efficient Fault Diagnosis Algorithm for Symmetric Multiple Processor Architectures," *IEEE Transactions on Computers*, Vol. C-27, No. 11, pp. 1059-1063, November 1978.

[Nakajima 1981] K. Nakajima, "A New Approach to System Diagnosis," *Proceedings of the 19th Allerton Conference on Communication, Control and Computing*, pp. 697-706, 1981.

[Pease *et al.* 1980] M. Pease, R. Shostak, L. Lamport, "Reaching Agreement in the Presence of Faults," *Journal of the ACM*, Vol. 27, No. 2, pp. 228-234, April 1980.

[Preparata *et al.* 1967] F. Preparata, G. Metze, R. Chien, "On the Connection Assignment Problem of Diagnosable Systems," *IEEE Transactions on Electronic Computers*, Vol. EC-16, No. 6, pp. 848-854, December 1967.

[Rabin 1983] M. Rabin, "Randomized Byzantine Generals," *Proc. 24th Symposium on Foundations of Computer Science*, pp. 403-409, November 1983.

[Raghavan and Tripathi 1991a] V. Raghavan, A. Tripathi, "Improved Diagnosability Algorithms," Vol. 40, No. 2, pp. 143-153, February 1991.

[Raghavan and Tripathi 1991b] V. Raghavan, A. Tripathi, "Sequential Diagnosability is Co-NP Complete," Vol. 40, No. 5, pp. 584-595, May 1991.

[Rangarajan and Fussell 1988] S. Rangarajan, D. Fussell, "A Probabilistic Method for Fault Diagnosis of Multiprocessor Systems," *Eighteenth International Symposium on Fault-Tolerant Computing*, pp. 278-283, June 1988.

[Rangarajan *et al.* 1990] S. Rangarajan, D. Fussell, M. Malek, "Built-In Testing of Integrated Circuit Wafers," *IEEE Transactions on Computers*, Vol. 39, No. 2, pp. 195-204, February 1990.

[Rangarajan and Fussell 1991] S. Rangarajan, D. Fussell, "Probabilistic Diagnosis Algorithms Tailored to System Topology," *Twenty-First International Symposium on Fault-Tolerant Computing*, pp. 230-237, 1991.

[Raynal 1988] M. Raynal, *Distributed Algorithms and Protocols*, pp. 137-163, John Wiley & Sons Ltd., 1988.

[Russell and Kime 1975a] J. Russell, C. Kime, "System Fault Diagnosis: Masking, Exposure, and Diagnosability Without Repair," *IEEE Transactions on Computers*, Vol. C-24, No. 12, pp. 1155-1161, December 1975.

[Russell and Kime 1975b] J. Russell, C. Kime, "System Fault Diagnosis: Closure and Diagnosability with Repair," *IEEE Transactions on Computers*, Vol. C-24, No. 11, pp. 1078-1088, November 1975.

[Scheinerman 1987] E. Scheinerman, "Almost Sure Fault Tolerance in Random Graphs," *SIAM Journal on Computing*, Vol. 16, No. 6, pp. 1124-1134, Dec. 1987.

[Schmeichel *et al.* 1988] E. Schmeichel, S. Hakimi, M. Otsuka, G. Sullivan, "On Minimizing Testing Rounds for Fault Identification," *Eighteenth International Symposium on Fault-Tolerant Computing*, pp. 266-271, 1988.

[Schneider 1984] F. Schneider, "Byzantine Generals in Action: Implementing Fail-Stop Processors," *ACM Transactions on Computer Systems*, Vol. 2, No. 2, pp. 145-154, May 1984.

[Schneider 1990] F. Schneider, "Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial," *ACM Computing Surveys*, Vol. 22, No. 4, pp. 299-319,

December 1990.

[Sengupta and Dahbura 1989] A. Sengupta, A. Dahbura, "On Self-Diagnosable Multiprocessor Systems: Diagnosis by the Comparison Approach," *Nineteenth International Symposium on Fault-Tolerant Computing*, pp. 54-61, June 1989.

[Shamir 1979] A. Shamir, "How to Share a Secret," *Communications of the ACM*, Vol. 22, pp. 612-613, 1979.

[Smith 1979] J. Smith, "Universal System Diagnosis Algorithms," *IEEE Transactions on Computers*, Vol. C-28, No. 5, pp. 374-378, May 1979.

[Somani *et al.* 1987] A. Somani, V. Agarwal, D. Avis, "A Generalized Theory for System Level Diagnosis," *IEEE Transactions on Computers* , Vol. C-36, No. 5, pp. 538-546, May 1987.

[Somani and Agarwal 1989] A. Somani, V. Agarwal, "Distributed Syndrome Decoding for Regular Interconnected Structures," *Nineteenth International Symposium on Fault-Tolerant Computing*, pp. 70-77, June 1989.

[Sullivan 1984] G. Sullivan, "A Polynomial Time Algorithm for Fault Diagnosability," *Twenty-Fifth Symposium on the Foundations of Computer Science*, pp. 148-156, 1984.

[Sullivan 1988] G. Sullivan, "An $O(t^3 + |E|)$ Fault Identification Algorithm for Diagnosable Systems," *IEEE Transactions on Computers*, Vol. 37, No. 4, pp. 388-397, April 1988.

[Tandem 1989] *Concepts and Facilities*, part no. 21620, Tandem Computers, Incorporated, March 1989.

[Toueg *et al.* 1991] S. Toueg, K. Perry, T. Srikanth, "Fast Distributed Agreement," *SIAM Journal of Computing*, Vol. 16, pp. 445-458, 1987.

[Turpin and Coan 1984] R. Turpin, B. Coan, "Extending Binary Byzantine Agreement to Multivalued Byzantine Agreement," *Information Processing Letters*, Vol. 18, pp. 73-76, February 1984.

[Xu 1991] J. Xu, "The $t/(n-1)$-Diagnosability and Its Applications to Fault Tolerance," *Twenty-First International Symposium on Fault-Tolerant Computing*, pp. 496-503, 1991.

[Yang and Masson 1985a] C. Yang, G. Masson, "A Fault Identification Algorithm for t_i-Diagnosable Systems," *Fifteenth International Symposium on Fault-Tolerant Computing*, pp. 78-83, June 1985.

[Yang and Masson 1985b] C. Yang, G. Masson, "A Generalization of Hybrid Faulty Diagnosability," *Fifteenth International Symposium on Fault-Tolerant Computing*, pp. 36-41, June 1985.

[Yang and Masson 1986] C. Yang, G. Masson, "An Efficient Algorithm for Multiprocessor Fault Diagnosis Using the Comparison Approach," *Sixteenth International Symposium on Fault Tolerant Computing*, Vienna, Austria, pp. 238-243, 1986.

[Yang and Masson 1987] C. Yang, G. Masson, "A New Measure for Hybrid Fault Diagnosability," *IEEE Transactions on Computers*, Vol. C-36, No. 3, pp. 378-383, March 1987.

[Yang and Masson 1988] C. Yang, G. Masson, "A Distributed Algorithm for Fault Diagnosis in Systems with Soft Failures," *IEEE Transactions on Computers*, Volume 37, No. 11, pp. 1476-1480, November 1988.

[Yang *et al.* 1986] C. Yang, G. Masson, R. Leonetti, "On Fault Isolation and Identification in t_1/t_1-Diagnosable Systems," *IEEE Transactions on Computers*, Vol. C-35, No. 7, pp. 639-643, July 1986.