The Consensus Problem in Fault-Tolerant Computing

MICHAEL BARBORAK AND MIROSLAW MALEK

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

ANTON DAHBURA

Motorola, Inc., Cambridge Research Center, One Kendall Square, Bldg 200, Cambridge, MA 02139

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 parallel and distributed systems and will impact the design of 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: Byzantine agreement, consensus problem, decision theory, processor membership, system diagnosis

1. 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 among the fault-free members of the resource population on a quantum of information in order to maintain the performance and integrity of the system. Such an agreement may be made on the configuration of the system, the synchronization of its clocks, the contents of its communications, or any other value requiring global consistency. The proposed approach is to diagnose and/or contain faults at the system level which will facilitate the consensus process. That is, this paper looks at general techniques for reaching agreement independent of the data being manipulated. Work in the area has increased with the proliferation of distributed systems that range from

M. Malek was supported in part by ONR contract N00014-88K-0543, ONR grant N00014-91-J-1858, NASA grant NAG9-426, and by IBM Agreement 203. M. Barborak was supported in part by a Microelectronics and Computer Development Fellowship.

CONTENTS

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 25 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. [1967] and on the *Byzantine Generals Problem* introduced by Lamport et al. [1982]. 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, followed by conclusions and summary in Sections 9 and 10.

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. Instead of looking at separate algorithms for each of these tasks, though, this paper surveys general techniques for agreement.

A distributed operating system shows the abundant need for consensus procedures. Figure 1 shows a general layered approach to fault management in which higher layers are dependent on lower layers to produce a fault-tolerant system from a basic system consisting of a group of processors connected by some unreliable communication network [Malek 1991]. A similar approach is presented by Cristian [1990]. The synchronization layer uses time to allow processors to recognize untimely messages, to detect faulty processors, and to order timely messages in implementing reliable communications. Reliable communications let fault-free processors pass fault-free messages that are used to either diagnose or mask faulty processors in order to agree on a correct sequence of computations and a correct result. Finally, the ability to agree on a diagnosis allows the fault-free processors to consistently reconfigure after a fault.

What must be recognized is that each of these layers is a separate consensus problem. First, the synchronization level maintains a global timepiece which is

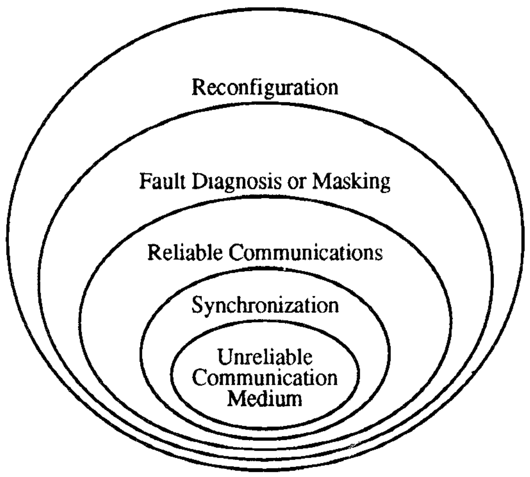


Figure 1. Consensus problems in fault management.

simply a consensus of all the fault-free PEs on a particular time value and a rate of change of that value. Second, using the results of the synchronization layer, 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 [Cristian et al. 1986]. Third, using the reliable communications facility, a consensus on the diagnosis of the system may be formed by the fault-free PEs. Finally, based on the diagnosis of the system, 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 Figure 1 consists entirely of consensus procedures.

In turn, this operating system may be used to implement fault tolerance with the state machine approach [Schneider 1990; Cristian 1991a]. Here, a service is a deterministic process or state machine whose execution is based solely on the reception of a message and which occasionally produces output. A fault-tolerant service is created by replicating the desired server and the service requests of its clients. 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, a client, or any other destination for a service output, must be able to determine a correct result from the many results produced by the service population. Thus, the client uses a consensus procedure to uncover the correct result, i.e., the client provides a voting service to its servers and itself.

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 the 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 NMR with a perfect voter. In other words, diagnosis allows consensus to be performed in what is assumed to be a fault-free environment. Researchers in the field of *system diagnosis* have explored solutions of this type for over 25 years [Preparata et al. 1967], and the results are applicable to wafer scale integration [Somani and Agarwal 1992; Rangarajan et al. 1990], large, loosely-coupled, distributed computer networks [Kuhl and Reddy 1980a], tightly-coupled multiprocessors [Dal Cin 1982], and to other kinds of multicomputer systems. Surveys on system diagnosis may be found in Dahbura [1988], Friedman and Simoncini [1980], Kreutzer and Hakimi [1987], Kime [1986], and 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 action. Therefore, NMR techniques may still be needed when any recovery procedure would be too costly. But implementation of NMR requires a voting mechanism that coalesces the n results into a single output. Obviously, the reliability of any process that uses this output is directly related to the reliability of the voting mechanism. If a single process is using this result, then it is sufficient for it to act as its own voter as the voting process will fail exactly when the process fails. But when (1) multiple processors rely on the output of the NMR system and (2) their subsequent computations must be consistent, then every processor must be able to agree on the output of the NMR system. In this case, a single point of failure is unacceptable, and the voting mechanism must be distributed. Work on the *Byzantine Generals Problem* (BGP) or *Byzantine agreement* explores the consensus problem given the need to distribute the voting process. A survey may be found in Raynal [1988] while general overviews of the consensus problem may be found in Turek and Shasha [1992] and Simons and Spector [1990].

Despite their different characteristics, the Byzantine agreement and system diagnosis problems have very similar goals, namely to produce a correct agreement despite a number of faults. That is, given a distributed system subject to failures, methods are needed to allow the system to progress in a consistent manner. Specifically, such methods may provide synchronization or reliable communication, but generally they allow the fault-free processor population to reach conclusions on the state of the system. On the one hand, system diagnosis does this by identifying faulty processors in order that their impact may be avoided, and on the other hand, Byzantine agreement does this with protocols that will mask any possible impact of the faulty processors. Yet in spite of their similarities in goals, 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. This ability to agree on the state of the system allows the fault-free processors to make correct and consistent progress. 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! An analysis of the effects of the PMC approach on processor availability, diagnosability, and utilization may be found in Dal Cin [1978; 1980] and Dal Cin and Dilger [1981].

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 directed edge (A, B) denotes that A tests B. Each edge is labeled with a 0(1) if the corresponding 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.

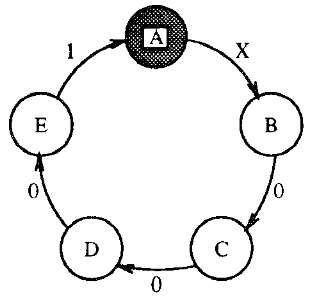
($A3$) A faulty processor produces unreliable test results.

($A4$) 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 and analyzed with the result distributed back to the system by a centralized supervisor be it a single PE or an NMR group of PEs [Blount 1978]. This is costly in terms of time, message passing, and system reliability, which is directly related to the reliability of this 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. $A4$ may exclude many practical fault situations.

Figure 2 shows an example of system diagnosis using the assumptions of the PMC model. A through E are processors where an edge (v_1, v_2) represents a test by v_1 on v_2. A label on an edge is the result of that test. The table in the figure relates the status of v_1 and v_2 to the result of the test of v_1 on v_2. In the figure, A is faulty as denoted by its gray color. The "X" on the edge (A, B) means that this result may be a one or a zero according to $A3$ without affecting the result of the diagnosis. 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].

Preparata et al. [1967] 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.



Status of Tester PE	Status of Tested PE	Test Result
Fault-Free	Fault-Free	0
Fault-Free	Faulty	1
Faulty	Fault-Free	X
Faulty	Faulty	X

Figure 2. Example of the PMC model.

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 Figure 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 A is repaired, 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. [1967] 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 [1974] showed that for the special case when no two processors test each other, these necessary conditions are also sufficient for *t-diagnosability*. 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 [1974] gave a general solution for one-step t-diagnosable networks. 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. Fujiwara and Ozaki [1979] characterized systems in which some PEs cannot test other PEs and in which some PEs may test themselves. Huang et al. [1989] characterized sequentially t-diagnosable systems.

Somani et al. [1987] provided a generalized characterization theorem that provides necessary and sufficient conditions for a particular fault set of any size to be uniquely diagnosable under symmetric and asymmetric invalidation fault models and with or without intermittent faults. (See Section 3 for descriptions of symmetric and asymmetric invalidation fault models and intermittent faults.) As a result, a particular syndrome in a t-diagnosable system with more than t faults may still be useful if it meets the necessary and sufficient conditions given by Somani et al. to be uniquely diagnosable. In other words, t-diagnosability only represents the performance of a diagnosis algorithm in the worst case.

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]. 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 [1991a] improved the efficiency of the t-diagnosability algorithm to $O(nt^{2.5})$. 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 [1978] 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. 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 [1988] 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.

Otherwise, the best solution in terms of worst-case efficiency is given by Dahbura and Masson [1984a]. They presented an $O(n^{2.5})$ 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. [1985a] 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. Even for larger values of n, the KTA algorithm performs more efficiently on average than the method given by Dahbura and Masson [1984a; 1984b]. The KTA scheme guesses 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. [1967] 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 $m = 1, 2 \dots, t$. They showed that if δ and n are relatively prime, then the system is one-step t-diagnosable. Meyer and Masson [1978], Mallela [1980], and Chwa and Hakimi [1981b] all gave $O(nt)$ solutions to the case of $\delta = 1$. (Note that the characterization of t-diagnosable systems makes nt the minimum value of E [Hakimi and Amin 1974].) Maheshwari and Hakimi [1976] described the Z_T systems; Chwa and Hakimi [1981b] gave the $D(n, t_0, X)$ class; and Dahbura et al. [1985b] defined a group of "self-implicating" structures, all of which have $O(|E|)$ diagnosis algorithms. Sullivan [1984] 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. [1985b].

Researchers have refined and detailed the model given by Preparata et al. [1967] 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

Wensley et al. [1978] were set with the task of designing a provably reliable aircraft control system, the result of which was known as Software Implemented Fault Tolerance (SIFT). A particular problem that faced them was that of clock synchronization in the presence of a single faulty clock. In order to guarantee a degree of reliability, only clock characteristics that could be *proven* to be sufficiently improbable could be ignored, and as a result, nothing was assumed about the behavior of a faulty clock. The authors proved that if all fault-free clocks should agree on the same synchronized time or even an approximate synchronization, then in the presence of a single faulty clock there would have to be at least three fault-free clocks to mask any of the effects of the faulty clock. An algorithm was given that performed fault-tolerant synchronization for this situation. In general, they reported that the number of clocks n must be greater than three times the number of faulty clocks t, i.e., $n > 3t$.

This work begun in the SIFT project was presented with more detail and generality in Pease et al. [1980]. The problem becomes one of general agreement among n processors, t of which could be faulty and behave in arbitrary manners. Once again, it was proven that n must be greater than $3t$. In this situation, each processor holds a secret value that it wishes to share with the other processors. The authors assumed that any two PEs had direct communication across a network which was (1) not affected by the failure of connected processors, (2) not prone to failure itself, and (3) subject to negligible delay. The sender of a message is identifiable by the receiver. It was also assumed that the system was synchronous. Without some sort of synchronization, Fischer et al. [1985] proved that consensus under this arbitrary fault model is impossible even if only one processor crashes during the protocol. Unlike the original work done by Preparata et al. [1967] on system diagnosis, described in Section 2.1, no central resources are assumed. The goal was to achieve *interactive consistency* made up of the following two conditions:

Consistency. Each fault-free processor should form an identical vector of values whose ith element corresponds to the ith processor in the system.

Meaningfulness. A vector element corresponding to a fault-free processor should be the actual secret value of that processor.

As an example of their solution, consider Figure 3 showing a system of four processors of which A is faulty as denoted by its gray color. A line between two processors represents a bidirectional private communication channel. Assume that each processor wants to share its secret value, and thus the consensus algorithm given in Pease et al. [1980] is initiated. Agreement is reached after two rounds of message passing. In the first round, the processors exchange their private values, e.g., B sends messages to processors A, C, and D telling them its value. 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, e.g., B sends a message to A with the values it received from C and D, to C with the values it received from A and D, and to D with the values it received from A and C. Now every processor has three numbers for the secret value of each other's PE, e.g., B has received values for A directly from A, via C and via D. If two of the three values for a particular PE are the same then it is used; otherwise a default value is used.

To show that interactive consistency is reached, first note that if A were fault free then B, C, and D would all receive A's correct value from A as well as from each other. Thus, interactive consistency holds. If A were faulty, then B, C, and D would only be required to agree on the same value for A, perhaps the default value. If the faulty processor A sent no messages then all the fault-free processors would record the default value, and the algorithm would be done. For a fault-free processor, say B without loss

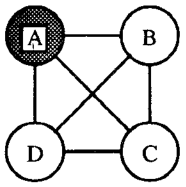


Figure 3. Example of Byzantine agreement.

of generality, to record a value v other than the default value for A, it must receive two reports of the same value from a combination of A and the other two fault-free processors C and D. If those reports came from C and D then those processors must have each received two reports of v for A, e.g., C received the same report of v from A and D. Then in this case, C and D will record v for A's value, and interactive consistency will have been reached. In the other case, the reports of v that B received came from A and one fault-free processor, say C without loss of generality. Therefore, D must have received v from B and C and thus records v for the value of A. B and C received v from every processor other than D, and therefore they record v for the value of A. Once again, interactive consistency is achieved [Pease et al. 1980].

Lamport et al. [1982] then cast the problem into a situation in which a Byzantine commanding general, who has surrounded the enemy with his many armies each led by a lieutenant general, wishes to organize a concerted plan of action, i.e., to attack or to retreat. Unfortunately, the Byzantine corps of generals has been infiltrated by traitors who want to influence this plan to the enemy's advantage. Despite this, the loyal Byzantine lieutenant generals must all reach the same conclusion either to attack or to retreat by sending messages back and forth among themselves. Moreover, their conclusion must agree with the commanding general's order if he, as per history of Byzantium, is loyal. An algorithm which completes this problem successfully is said to reach *Byzantine agreement*.

The Byzantine generals are replaced by processing elements in a distributed computing environment. A single PE, representing the commanding general, has a secret value from a set of ordered values that it wishes to broadcast to every other processing element. In the simplest case, this value may be a single binary digit as Turpin and Coan [1984] showed that any value representable by k bits could be agreed upon by performing k iterations of the consensus algorithm, one iteration for each bit. Faulty PEs may work in collusion to try to break the agreement by sending inconsistent information to different processors. Once again, it is shown that $n \geq 3t + 1$ must hold even in the case of only needing to reach approximate agreement.

The problem is somewhat different than the one studied in Pease et al. [1980] in that only one PE is interested in sharing its secret value as opposed to all of the members sharing their values. By assigning the role of commanding general to every PE and running the resulting consensus algorithms concurrently, the problems become identical. Because of the similarities of the problems both will be described as Byzantine Generals Problems with specification given as needed.

Unlike the system diagnosis algorithms in which arbitration of conflicting test results is NP-complete except for special cases, the Byzantine agreement procedure can resolve conflicting values by simply taking a deterministic majority vote of the ordered 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 example and requires $t + 1$ rounds to complete (Fischer and Lynch [1982] showed that at least $t + 1$ rounds are needed for all deterministic solutions to the Byzantine Generals Problem) [Lamport et al. 1982]. Unfortunately, the message size grows exponentially at each round ($O(n^{t+1})$). Dolev and Reischuk

[1985] gave a lower bound of $O(nt)$ on the number of messages required to reach agreement, and algorithms have been given that meet this bound. Work on more efficient BGP algorithms may be found in Section 5.2.2.

Lamport et al. [1982] and Dolev [1981; 1982] examined the connectivity requirements needed for reaching Byzantine agreement. To this point, it had been assumed that the group of processors was completely connected to allow for private communication between any pair of processors. Lamport et al. gave an algorithm that works for systems with *3t-regular* system topologies. (*3t-regular* implies that each processor is connected to $3t$ other processors in such a way that there is a path between any processor p and the neighbor of any other processor q such that that path does not pass through q and that any pair of paths between p and the neighbor of q have no nodes in common.) Dolev gave a weaker requirement by showing that the connectivity of the communication graph must be at least $2t + 1$. He also noted that reducing connectivity will most likely result in more rounds required for Byzantine agreement.

Byzantine agreement becomes much simpler if messages are *authenticated* or *signed* [Pease et al. 1980; Lamport et al. 1982]. A message is *authenticated* if: (1) a message signed by a fault-free PE is unable to be forged; (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 situation, 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 [1985] gave an algorithm that uses authenticated messages that requires $O(t)$ rounds and $O(n + t^2)$ messages, which they prove is a lower bound. In practice, cryptography techniques may be used for authentication if the associated overhead is not prohibitive (Dolev and Reischuk [1985] proved that at least $O(nt)$ signatures would be required to reach agreement). In other cases, techniques as simple as parity or checksums may provide the needed reliability depending on the failure semantics of the processors.

With a system diagnosis algorithm, every processor must pass a trial of tests. If it passes, then its output is assumed to be correct until the next round of tests. In this way, every processor may operate on its own set of jobs, and a consensus on the results is formed with simple message passing as it is assumed that the environment is fault free until testing uncovers a fault. With a Byzantine agreement protocol, 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. This masking process is performed via consensus, and therefore no extra steps are required to share the system state. At a high level, the two solutions are striving for identical behavior. That is, a multiprocessor is given a set of inputs, and despite any failures, it is returning a correct set of outputs based on its ability to reach consensus. The difference lies in the performance of the two algorithms in achieving this goal as a result of the use of fault detection in system diagnosis and fault masking in Byzantine agreement. The system diagnosis solution should operate with a high throughput until a fault is detected, possibly resulting in an incorrect output prior to detection, and recovery restores the system performance. On the other hand, the Byzantine agreement protocol should perform with extremely high reliability, and with a consistent, though lower, throughput until the number of faulty processors makes it impossible to mask out their results. 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 as these characteristics often determine the efficiency of the consensus algorithms.

3. THE FAULTY ELEMENT

Knowing how a processing element fails is key to making realistic assumptions and creating workable algorithms to detect and mask the faulty PE. Careful examination of the characteristics of faulty processors has resulted in the proposition of many fault models, the effect of which has been a wide range of algorithms between and within the areas of system diagnosis and Byzantine agreement. 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 and Classes

A fault model must define the behavior of a PE once it has become faulty. Ideally, the faulty processing element will behave with as much predictability and beneficence as a fault-free PE, but generally this is not the case. For system diagnosis, the fault model is a description of the test results given the status of the tester and the tested; for the Byzantine Generals Problem, it is a description of the limitations of a faulty processor. Of course, there is no reason why one fault model should be used for system diagnosis and another for Byzantine agreement, but with little exception, this division has been maintained by researchers.

3.1.1 A Failure in System Diagnosis

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.* Figure 4 shows some other proposed test result models. Barsi, Grandoni, and Maestrini introduced a model (known from now on as the BGM model) in Barsi et al. [1976] in which a faulty processor would always test faulty regardless of the condition of the testing processor. Given a large number of test stimuli, it may be reasonable to assume 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 of the BGM model is known as *asymmetric invalidation.* Kreutzer and Hakimi [1983] extended this assumption by proposing that a faulty tester would always report a nonfaulty PE as being faulty. The HK Model 1 and HK Model 2 are called *reflexive* and *irreflexive invalidation*, respectively. The result of restricting the PMC model with the BGM or HK models is a limitation on the possible syndromes that can occur, theoretically making it easier to identify which PEs are actually faulty.

A diagnostic procedure must take into account the possible fault classes prevalent during the testing process. That is, a faulty processor may or may not exhibit a faulty behavior depending on the fault class of that PE. In system diagnosis, researchers have examined the effect of classifying processor faults as *transient*, *intermittent*, or *permanent*. Results may be found in Section 5.1.3. 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 software or hardware is faulty. By its nature, an intermittent fault will not occur consistently, which makes its diagnosis a probabilistic event over time. One effect of this can be fault syndromes that are not compatible with the assumptions made by Preparata et al. [1967] (discussed in Section 2.1) making diagnosis more difficult. Permanent or *hard* faults are software or hardware faults that always produce errors when they are fully exercised.

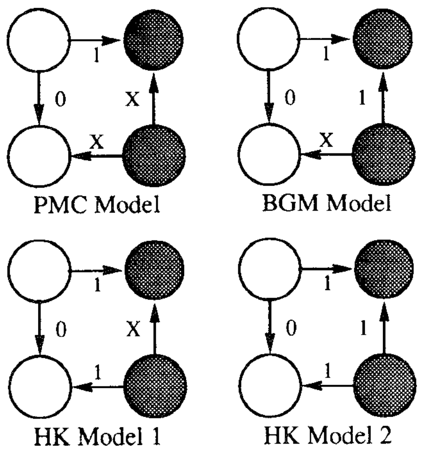


Figure 4. Test validity models. A directed edge denotes a test by a PE on another PE. A 0 (1) denotes a pass (fail). An X indicates that the PE will produce an unreliable result after performing the test. Faulty PEs are shaded

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 transient 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.

3.1.2 A Failure in Byzantine Agreement

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 [1985] 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. As in system diagnosis, limiting the fault model simplifies the solution [Lamport et al. 1982].

Defining algorithms that work only for this adversary model can be limiting and impractical. Therefore, another classification of faults, which may be just as realistic for a given application, has been adopted by this community. This taxonomy divides processor faults into various groups with the interesting property that a stronger class is a subset of a weaker class. The classes, from strongest to weakest, are *fail-stop* faults, *crash* faults, *omission* faults, *timing* faults, *incorrect computation* faults, and *Byzantine* faults. Figure 5 shows a graphical representation of the subsetability of these fault classes.

Fail-Stop Fault: The fault that occurs when a processor ceases operation and alerts other processors of this fault [Schlichting and Schneider 1983].

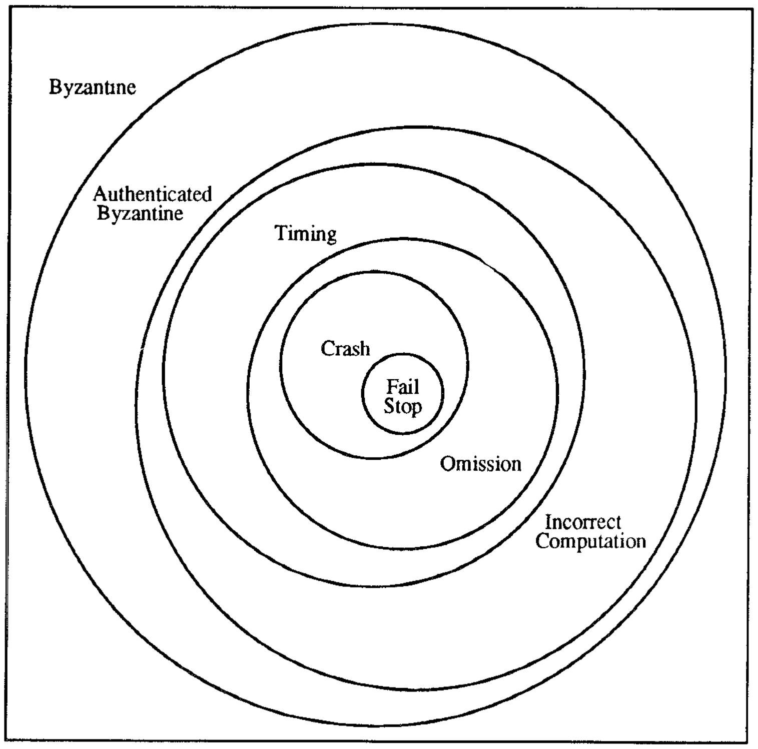


Figure 5. An ordered fault classification.

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 [Cristian et al. 1986]. In particular, a *send omission fault* occurs when a processor fails to send a required message on time or at all, and a *receive omission fault* occurs when a processor fails to receive a required message and behaves as if it had not arrived.

Timing Fault: The fault that occurs when a processor completes a task either before or after its specified time frame or never [Cristian et al. 1986]. This is sometimes called a *performance fault.*

Incorrect Computation Fault: The fault that occurs when a processor fails to produce the correct result in response to the correct inputs [Laranjeira et al. 1991].

Authenticated Byzantine Fault: An arbitrary or malicious fault, such as when one processor sends differing messages during a broadcast to its neighbors, that cannot imperceptibly alter an authenticated message [Lamport et al. 1982].

Byzantine Fault: Every fault possible in the system model [Lamport et al. 1982]. This fault class can be considered the universal fault set.

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, omission, and timing 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 in the previous section that stress error detection in the data domain. As a result, both models may be applied concurrently for a more complete approximation of the behavior of a faulty PE. The effects of time domain testing may be seen in Section 5.1.7, which describes a set of processor membership protocols that test the fault-free status of a processor solely in the time domain.

3.2 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.

Defining the possible impacts of a fault within the confines of its fault model is desirable as a diagnosis or agreement algorithm can take advantage of the dependent nature of these faults. That is, the concurrence of multiple failures could be diagnosed as a single fault which is advantageous when the number of faults must be bounded. Conversely, it may be the case that the concurrence of two particular elements failing happens with such a small probability that the diagnosis of one as faulty implies the other one is fault-free. The next section discusses some work done in this area. For a study of fault impact in parallel-processor systems refer to Menezes et al. [1992].

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. Nonetheless, some conclusions about the status of a processor may be drawn from its actions during Byzantine agreement as discussed in Section 5.2.6.

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 [1980a] proposed 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.

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 1970; Russell and Kime 1975a; 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) which 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) which again is the case in the PMC model. The tests in a MIPT or MUPT environment may be simpler to write and quicker to execute, because fewer 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 [1976] characterized MIPT/MUPT systems while Holt and Smith [1981] examined their diagnosability and diagnosis.

A somewhat different model is examined by Gupta and Ramakrishnan [1987]. Here a processor is tested by a number of other processors, each able to completely test some portion of the functions of the tested processor. Therefore, to completely test a PE, the test results of many processors must be combined. Unlike the MIPT model described above, though, a single faulty tester does not invalidate the test results of the other testers. Such systems were characterized by the authors.

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 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 has 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 [1980] introduced the comparison approach in the context of system diagnosis and presented a method to assign comparison edges in the system graph. A syndrome of comparison results is created by labeling an edge with a zero if it connects two PEs that agree and with a one if it connects two that disagree. Malek assumed that faulty processors would never produce identical results. The resulting syndrome is diagnosed by a centralized supervisor as if it were the result of a PMC-modeled system. Chwa and Hakimi [1981a] proposed a similar comparison approach independently with the exception that faulty processors could possibly produce the same results. Maeng and Malek [1981] 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. 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.

As an example of the original comparison model given in Malek [1980], consider the four-processor system of Figure 6 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 Figure 6. The next step is to analyze this syndrome, for example, using the $O(n^{2.5})$ algorithm of Dahbura and Masson [1984a] described in Section 2.1. 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 fault free 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.

Ammann and Dal Cin [1981] examined the necessary conditions for a comparison-based system to be t-diagnosable. They showed that the minimum degree

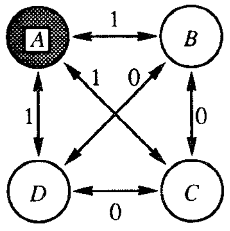


Figure 6. An example of comparison testing and diagnosis.

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

of the testing graph must be greater than or equal to t (implying that the number of tests must be greater than or equal to $\lceil nt/2 \rceil$) and gave conditions under which a processor could have this minimal degree. They also showed that $d_G \geq t + 1$, where d_G is the minimum degree of a processor in the testing graph, is sufficient for a system to be t-diagnosable. In general, finding the complete and correct set of faulty processors using the comparison model is NP-hard [Blough and Pelc 1992], but if the system is t-diagnosable, the problem is solvable in polynomial time. Ammann and Dal Cin gave an $O(n^2)$ sequential algorithm for a subset of the t-diagnosable systems as well as a parallel algorithm for the diagnosis of tree topology multiprocessors [Ammann and Dal Cin 1981; Dal Cin 1982]. Sengupta and Dahbura [1989] completed the characterization of comparison-based, t-diagnosable systems and gave a polynomial algorithm for the diagnosis of all t-diagnosable systems under the

Maeng/Malek model. Blough and Pelc [1992] gave efficient algorithms for diagnosis when the testing assignment is a bipartite graph.

In the data domain, i.e., regardless of the timing of outputs, the only way a processor may be faulty is if it produces an incorrect output. Therefore, regardless of the fault model employed, comparison testing can uncover faulty or suspicious processors, and diagnosis of Byzantine-failing PEs is possible. In fact, a close examination of the Byzantine agreement algorithm discussed in Section 2.2 reveals that comparison of results is the mechanism used to identify the correct result. Thus the tools for diagnosis based on comparison testing are already in place. Their exploitation is more carefully discussed in Section 5.2.6.

4.4 Time Domain Testing

An alternative fault classification largely based in the time domain was shown in Figure 5. This approach leads to the testing of processors with respect to time. 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. Thus timing errors may go undetected in a data domain fault model while data errors may go undetected in a time domain fault model. The processor membership algorithms discussed in Section 5.1.7 are largely based on time domain testing.

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 Lamport et al. [1982].

5.1 Extensions to the PMC Model

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

The existence of tests. Much system-level diagnosis research has assumed the ability of processor A to test processor B *completely*, i.e., with 100% coverage. In Section 4, the comparison approach was presented as a solution, but this still does not assure a complete test. Therefore, probabilistic techniques have been developed to cope with imperfect tests [Blount 1977; Dahbura et al. 1987; Blough et al. 1988].

Permanently faulty PEs. The only fault class Preparata et al. [1967] considered was the permanent-fault class, excluding the possibility of intermittent or transient faults. It was assumed that a faulty PE performed all assigned jobs incorrectly and that a faulty or fault-free PE would maintain its status during all testing procedures. To weaken these assumptions, work has been done for systems which suffer from intermittent faults [Mallela and Masson

1978; 1980; Yang and Masson 1985b; 1987].

The number of faults is bounded. Preparata et al. [1967] proved for their model that the number of PEs in the system must be greater than or equal to twice the number of faulty PEs plus one, i.e., $n \geq 2t + 1$. This bound has been used in the characterization and diagnosis of these systems, and the result has been costly diagnosis strategies. In practice, fault-tolerant systems handle at most one fault if the time to repair is much shorter than the time between failures. On the other hand, Wafer-Scale Integration, which is the placement of many processors on a single wafer, typically has a poor yield leading to $2t + 1 > n$. The solution to the problem of either overestimating or underestimating the number of faults in the diagnosis algorithm has been to make no assumptions about the number of faults. This might mean that the fault will not be correctly identified, but a sufficiently low probability of this happening can make these algorithms satisfactory.

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 [Saheban et al. 1979]. 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. [1978] uses a time-varying subset of PEs to perform the system tasks while the other PEs conduct testing and diagnosis. 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 are given a low priority for identification in the diagnosis 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 examination. 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 diagnostic 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 [Nair 1978; Kuhl and Reddy 1980a; 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 [1975] 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 is identifiable to within a set of at most s PEs. Karunanithi and Friedman [1977] looked at the effect of t/s-diagnosability on the diagnosis of certain network topologies.

A special and important case of set diagnosis is t_1/t_1-diagnosis that was introduced in Friedman [1975] and characterized by Chwa and Hakimi [1981b]. In such a system, even if at most t_1 PEs are faulty, diagnosis can locate a set of at most t_1 PEs that includes all of the faulty processors. A system might be t-diagnosable and t_1/t_1-diagnosable with $t \le t_1$. If $f \le t$, where f is the number of faults, then the fault set is obviously identifiable. If $t < f \le t_1$ then Yang et al. [1986] 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. 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 [1978] who noted that if $n \gg t_1$, then only $n\lceil(t_1 + 1)/2\rceil$ tests are needed to construct a t_1/t_1-diagnosable system. This is almost half the number of tests required by a t-diagnosable system where $t = t_1$ [Hakimi and Amin 1974].

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

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

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 to reliably test 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 [1991] examined $t/(n-1)$-diagnosability and its use in the diagnosis and repair of constant-degree systems as well as software fault tolerance. Optimal configurations for these systems were presented.

5.1.2 Adaptive Testing

Classically, the testing assignment for a system was determined prior to the execution of the diagnosis algorithm, and was left unchanged during execution. Nakajima [1981] saw that this was restrictive in that tests could be adapted as information was uncovered to optimize the speed and accuracy of the diagnostic process. He proposed that once a fault-free PE was located, it could be used to reliably test the other PEs in the system as long as it remained fault free. Assuming this approach, Blecher [1983] showed that, in the worst case, at least $n + t - 1$ tests are required to identify all t faulty units for $n \ge 3$ which is many fewer tests than the optimal nt tests required for fixed testing assignments. As an example, the adaptive testing algorithm given by Hakimi and Nakajima [1984] 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. 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].

Pelc [1992] presented a number of algorithms based on comparison testing to perform t-fault detection, sequential t-fault diagnosis, and one-step t-fault diagnosis in both adaptive and nonadaptive manners. In every case, the nonadaptive algorithm required more tests than the adaptive case. (In Section 7.3, implementations of adaptive testing schemes at Carnegie Mellon University are described.)

5.1.3 Intermittent Faults

Mallela and Masson [1978] were the first to include intermittent faults in their system model. 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 fails, 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*, 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. Up to this point, only systems with reliable communication links were considered. That is, the models assumed that faults only occurred within the PEs. Therefore, Yang and Masson [1988a] introduced another diagnosability measure known as (t_h/t_{hi}-unit; σ-link)-diagnosability which describes a t_h/t_{hi}-diagnosable system in which as many as σ test outcomes may be altered by unreliable communication links. The respective authors that introduced each of those diagnosability measures also have characterized the systems described by these 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 diagnostic algorithms are no longer directly applicable. Dahbura and Masson [1983a] introduced the idea of *greedy diagnosis* to identify faulty processors from a pf-incompatible syndrome. 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, in the most general case they showed that the problem is NP-complete. Therefore, Kozlowski and Krawczyk [1991] gave necessary and sufficient conditions for the comparison assignments to achieve correct and complete diagnosis and gave an $O(n|E|)$ diagnosis algorithm for such systems.

Yang and Masson [1985a] 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).

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, we 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 [1976] assigned a reliability to each PE in the network. 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 [1986] generalized the $O(n^{2.5})$ diagnosis algorithm of Dahbura and Masson [1984a] (see Section 2.1) for use with p-t-diagnosable networks. Sullivan [1987] gave a polynomial algorithm ($O(n^4)$) which approximates the p-t-diagnosability of a testing graph to within an additive factor. He showed that finding the exact diagnosability is NP-hard.

Blount [1977] took a different approach to probabilistic diagnosis by assigning a probability of correctness to each test rather than to the PEs themselves. Unlike Preparata et al. [1967] 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 author extended this model to include the syndrome analysis process. That is, Blount realized that the syndrome-analyzing supervisor in the PMC model would most likely be implemented by a "committee" of processors drawn from the pool of processors being diagnosed. Obviously, under this model, there would be a probability that committee members would be faulty resulting in an incorrect syndrome analysis. Blount [1978] studied the probability of correct diagnosis given these assumptions.

The general problem is to diagnose a system that suffers from intermittent failures and that has tests with imperfect coverage. Dahbura et al. [1987] were the first to examine this problem using probabilistic diagnosis under the comparison approach. 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. [1967], 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. Using a similar model, Pelc [1991] and Blough and Pelc [1992] proved that finding a given testing assignment's diagnosability and performing optimal probabilistic diagnosis, i.e., finding the most likely set of faulty processors, are NP-hard problems. They also showed that the deterministic comparison model given by Malek [1980] is a limit of their probabilistic model and that diagnosis for the Malek model in arbitrary systems is also NP-hard.

Blough et al. [1988] reexamined the problem based on the more conventional approach of one PE testing another. They assigned a reliability to the processing elements and a coverage probability to the tests. It was shown that to perform correct diagnosis of $O(n)$ faults 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. The authors 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 as n approaches infinity. In Blough et al. [1992a] the number of required tests was improved to $O(n \log n)$.

Blough et al. [1989] continued their work to reduce the number of required tests which in turn reduces the need for physical communication paths thus allowing diagnosis for a large number of faults in such systems as hypercubes. It was shown that their probabilistic diagnosis algorithm is almost surely correct for a class of regular systems that includes the hypercube while, in general, regular systems of degree $O(\log n)$ are not diagnosable with high probability. Scheinerman [1987] examined a similar system model, but an error invalidates his lower bound on the number of required tests. Blough et al. [1989] showed that for a class of systems in which a set of processors acts as testers for the entire system, diagnosis would be correct with probability approaching one as long as the number of tests grew just faster than n. They extended their results showing that diagnosis could be correct with a probability arbitrarily close to one for systems performing a linear number of tests [Blough et al. 1992b]. They also proved that this is the best possible result since a sublinear number of tests must result in a probability of correct diagnosis approaching zero.

Fussell and Rangarajan [1989] and Rangarajan and Fussell [1988] used probabilistic diagnosis in a distributed environment to assure, with high probability, correct diagnosis in a system with arbitrary connectivity. This result is similar to that of Blough et al. [1989] who showed that correct diagnosis could be attained in systems with constant connectivity. (Berman and Pelc [1990] used the probabilistic model of Blough et al. in a distributed environment.) 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 [1989] accomplished the reliable diagnosis of sparsely connected networks by comparing processor results across several tasks. 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 [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 [1991] adapted their algorithm to tailor itself to any system topology. 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* and *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.

5.1.5 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 the relaying of the diagnostic 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 [1988] discussed this quandary. 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.

Nair [1978] provided the first formulation of the system diagnosis problem in a distributed environment. His concept was a *roving-diagnosis* approach in which only a subset of the system's PEs performs diagnosis at any time, creating an entire system diagnosis as the subset "roves" over the entire set of PEs.

Smith [1979] gave simple system diagnosis algorithms that do not use a centralized observer and that 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.

Ciompi et al. [1981] saw the problem as one of distributing the centralized analysis of the syndrome. In their MuTeam approach, testing was carried out as required for t-diagnosability, but once this was completed, results were not sent to a centralized observer, but rather were disseminated using a consensus protocol similar to Byzantine agreement. Then, each PE would 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. As a result, techniques resulting in dynamic diagnosis have been explored.

Kuhl and Reddy [1980a] introduced the term *distributed diagnosis*. In their model, 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. Dal Cin [1984] and Dal Cin and Florian [1985] examined the problem of distributed diagnosis based on comparison testing as well as time domain testing.

Kuhl and Reddy [1980a; 1981], joined later by Hosseini et al. [1985], presented a series of **SELF** algorithms. **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 jth 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 link 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, (1) the information about Q is saved in the fault vector and (2) 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 1980a]. Pradhan and Reddy [1982] independently proposed a similar scheme of *test and accept* in order to reliably pass diagnosis information in a distributed system.

Bagchi and Hakimi [1991] gave an optimal algorithm for the system model of Kuhl and Reddy [1980a] that assumes no more than t faults and fault-free communication links. Their algorithm requires at most $n - 1 + p(t + 1)$, i.e., $O(n)$, diagnosis operations and $3n \log p + O(n + pt)$, i.e., $O(n \log n)$, messages from fault-free PEs where p is the number of fault-free PEs. Bagchi [1992] examined the same problem but for a specific interconnection, namely a hypercube, with the result being an algorithm requiring $O(n + t \log n)$ tests and $O(n + t^2 \log n + \log^2 n)$ messages, or $O(n)$ test and $O(n)$ messages if the number of faulty units t is small.

In **SELF3**, Kuhl and Reddy [1980b; 1981] 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. In the **Modified Algorithm SELF3**, Hosseini et al. [1985] altered **SELF3** such that fault-free PEs are never temporarily misdiagnosed as faulty. See also the **NEW_SELF** algorithm described in Hosseini et al. [1984].

The work of Hosseini et al. [1984; 1985] illustrates the trend toward more reliable accounting of the nuances of an actual system. Liaw et al. [1982] modeled a heterogeneous distributed system with a graph-theoretical model where processors are marked as *testing* or *nontesting* units. A testing unit is a processing element with the capability to test at least one other processor. A nontesting unit does not test any objects and relies on other PEs to give it diagnostic 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. [1985] used these ideas in their algorithm for nonhomogeneous distributed systems. The problem of link failure also may be examined as a routing problem. A discussion of this may be found in Bertsekas and Gallager [1987].

Lombardi [1985] took the NMR technique for fault masking and cast it into the system diagnosis framework: 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 earlier in this section [Ciompi et al. 1981].

Yang and Masson [1986] considered the distributed diagnosis of a t_l-diagnosable system. The soft-fail model they employed covers intermittent faults in both the PEs and communication links as long as the total number of faults does not exceed t_l. 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 fact, in a Byzantine manner. Therefore, the diagnosis is not and cannot be guaranteed to be complete. A set of maliciously faulty processors could postpone a complete diagnosis indefinitely.

This algorithm overlaps with work done by Dolev [1982] who studied the diagnosis of systems suffering from Byzantine failing processors and reliable communication links with his Crusader agreement algorithm described in Section 5.2.6. In the Yang and Masson algorithm [1988b], each processor begins by testing other processors according to a predetermined testing assignment. The results of these tests are needed by other processors to complete their test syndromes, and therefore the test information is sent to the other processors in the system. The trouble is that this must be done reliably, and the result is a protocol that is very similar to Crusader agreement, a form of Byzantine agreement. This protocol guarantees that fault-free processors can reliably transfer information, and, therefore, all the useful syndrome information, i.e., all the results of tests given by fault-free PEs, will be reliably received by all fault-free processors. At this point each processor performs diagnosis on the syndrome it has received according to the algorithm given by Yang and Masson [1985a] which can locate faulty processors given that the syndrome is pf-compatible (Section 5.1.3). The result is the diagnosis of a Byzantine failing system based on the ability of a test to uncover these failures.

5.1.6 Diagnosis of Multiprocessors

The diagnosis of tightly coupled multiprocessors presents some interesting problems in that the communication topology is often limited (particularly in wafer-scale integration), and the number of faulty processors may be very high relative to n. In general, any diagnosis algorithm is applicable to such a system, but the capability to perform the diagnosis in a parallel fashion is inviting. Ammann and Dal Cin [1981], Dal Cin [1982], and Somani and Agarwal [1992] have presented both sequential and parallel algorithms specifically targeted for such systems.

5.1.7 Processor Membership

The principle of treating a group of processors as a single identity in order to provide a fault-tolerant service was introduced as part of the ISIS system [Birman 1985]. A result of this approach is a need to know who is a member of the group, a need satisfied by *processor membership*.

Processor membership is similar to distributed diagnosis in that both must maintain a list of who is fault free without aid from a centralized observer. 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. Cristian et al. [1986] and Cristian [1989] presented three solutions given as overlays on a synchronous system with *atomic broadcasts*. The atomic broadcast assures that all or none of the fault-free processors will receive the message (*atomicity*), that every fault-free processor receives messages in the same order (*order*), and that the broadcast is completed in some known time bound Δ (*termination*). With this mechanism, Cristian [1991b] presented three protocols for processor membership: the *periodic-broadcast* protocol, the *attendance list* protocol, and the *neighbor surveillance* protocol. These protocols handle the cases of faulty processors leaving the membership and fault-free or repaired processors joining the membership.

In each of these three protocols, Cristian [1991b] 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 broadcasts a "present" message with some predetermined and globally known frequency. A new membership is created when a processor broadcasts a "new-group" message at time T, which 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 Figure 7 in which A has become faulty. Assume that the network 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 a time δ after the next scheduled round of broadcasts, and at that 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. In the case that the network is based on point-to-point communications, this overhead could be prohibitive. For broadcast-channel-based networks, though, the result is only an $O(n)$ message overhead [Cristian 1989]. 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 checks the timeliness of the attendance list and forwards it. If any of the members does not receive the list in time then an error has occurred, and a "new-group" request is made. The result is that 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

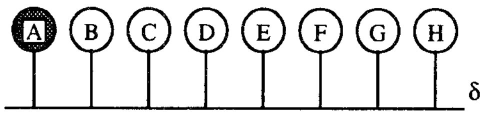


Figure 7. A broadcast bus network performing processor membership.

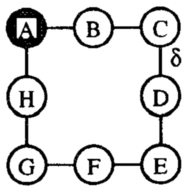


Figure 8. A point-to-point network performing processor membership.

maximum-departure detection time is proportional to the time it takes the attendance list to circulate through the membership [Cristian 1991b].

Consider Figure 8 in which the network uses point-to-point communication 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 elects itself to periodically initiate the roll call. As all the members have the same information on who is in the membership, a deterministic selection process known by each processor and performed independently of the other members assures that only one member will 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 similar to the distributed-system diagnosis problem as both strive to determine which processors are faulty and which are not. Whereas distributed diagnosis explicitly utilizes a test to do this, an implicit test is assumed in the processor membership model in that a PE is determined to be fault free if it can transmit and forward messages in a timely manner. Comparing these protocols given by Cristian [1991b] with the **Adapt** algorithm for distibuted diagnosis described in Section 7.3 leads to the observation that neither assumes an initial membership; neither assumes foreknowledge of the system topology; and neither assumes a simultaneous initiation of their respective algorithms [Stahl et al. 1992].

Where the differences between processor membership and distributed-system diagnosis lie is in the time assumptions relaxed for processor membership. That is, Cristian [1991b] has analyzed these protocols based on assumptions about approximately synchronized clocks and variable message delays. These time characterizations of the system along with the atomic-broadcast protocol allow the processors in the membership to maintain a *consistent* view of the membership, given that it can only be inconsistent for some determinable amount of time. But as well as a difference, this quality can be seen as an addition to distributed-system diagnosis. That is, the time domain fault model that Cristian uses is orthogonal to the fault models used in distributed diagnosis, and therefore it and Cristian's time-based analyses are applicable to distributed diagnosis without violating any previous assumptions. (Diagnosing faults in the time domain as well as the data domain for distributed-system diagnosis was proposed by Dal Cin and Florian [1985].) Conversely, the data domain fault detection and location abilities of distributed diagnosis are equally applicable to processor membership protocols which need to explicitly cover those types of faults.

5.2 Research on the Byzantine Generals Problem

The original Byzantine agreement algorithm presented by Pease et al. [1980] was expensive in both its communication and system requirements. 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.) It is also evident that if during an agreement algorithm a set of values received by a processor does not agree, then some processor must be faulty. Identifying these processors is also discussed in this section.

5.2.1 System Requirements

Fischer et al. [1985] gave the very important result that distributed, deterministic (see the later section on randomized algorithms) consensus is impossible in an *asynchronous* system with just one faulty processor. If no assumptions are made about the upper bound on how long a message may be in transit Δ, nor 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. [1987] showed that if either Δ or Φ were unbounded, then consensus is impossible in the case of one fault. Dwork et al. [1988] explored the effects of *partial synchrony*, bounding Δ and Φ individually, on Byzantine agreement and gave algorithms that operate on partially synchronous systems. Earlier work bounding Δ was done by Attiya et al. [1984]. It is worthwhile to note that the randomized Byzantine agreement algorithms described later can operate in an asynchronous system as they may take an infinite number of rounds to complete. The probability that this will happen, though, is zero [Bracha and Toueg 1985].

Dolev et al. [1987] identified five system characteristics that affect the ability to reach Byzantine agreement. These characteristics are as follows:

Processors. Either asynchronous, in which processors may wait an arbitrarily long but finite period between steps, or sychronous, in which after some processor has taken Φ steps every other processor has taken a step.

Communication. Either asynchronous, in which messages may take an arbitrarily long but finite period to be delivered, or synchronous, in which there is a constant $\Delta \geq 1$ such that messages are delivered within Δ steps.

Message Order. Either asynchronous, in which messages may be delivered out of order, or synchronous, in which messages are received in an order according to the real time that they were sent.

Transmission Mechanism. Either point to point, in which in an atomic step a processor may send a message to at most one processor, or broadcast, in which in an atomic step a processor may broadcast a message to all processors.

Receive / Send Either separate, in which in an atomic step a processor cannot both send and receive, or atomic, in which in an atomic step a processor will both receive and send.

Dolev et al. [1987] showed that n-resilient agreement, i.e., agreement in which n faulty processors may be tolerated, may be obtained in four cases, namely:

(1) Synchronous processors and synchronous communication
(2) Synchronous processors and synchronous message order
(3) Broadcast transmission and synchronous message order
(4) Synchronous communication, broadcast transmission, and atomic receive/ send.

They further showed that agreement may not be reached at all except in two other cases, asynchronous processors, synchronous communication, point-to-point transmission, atomic receive/send, and both asynchronous and sychronous message order, in which 1-resilient agreement may be had. Another presentation of this material may be found in Turek and Shasha [1992].

An interesting contrast to the need for synchronization of processors and communications is approximate agreement (described later). In this case, processors need only agree on a value to within some constant. Under this requirement, a synchronous environment is not required.

5.2.2 Efficient Byzantine Agreement

The algorithm given by Pease et al. [1980] 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. 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 with the total number of message bits being $O(t^2)$. 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 $O(t^{15})$ processors, $t + 1$ rounds of communication, and messages of size $O(t \log n)$ while Toueg et al. [1987] presented a scheme with $3t + 1$ processors, $2t + 1$ rounds, and message sizes polynomial in the number of processors.

Bar-Noy and Dolev [1991] asked if there even exists an algorithm that optimizes all three parameters. 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 on 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.

A method of hybridizing Byzantine agreement algorithms has been introduced to produce efficient results from existing algorithms that are optimal in certain, but not all, respects. Bar-Noy et al. [1992] base their hybrid-agreement protocol on three less efficient protocols: (1) an exponential algorithm similar to that presented in Pease et al. [1980] that can tolerate $\lfloor (n - 1)/3 \rfloor$ faults in just $t + 1$ rounds but requires that message size and local computation time grow exponentially; (2) a more efficient $\lfloor (n - 1)/4 \rfloor$ -resilient algorithm, and (3) an optimally efficient agreement algorithm, except in its resiliency, which is $\lfloor \sqrt{n/2} \rfloor$ faulty processors. The hybrid algorithm runs the first algorithm for a certain number of rounds, switches to the second algorithm for a certain number of rounds, and finally switches to the third algorithm to complete the Byzantine agreement. The number of rounds that each algorithm runs is based on the number of faulty processors that are detected. (See Section 5.2.6 for a discussion of fault detection during Byzantine agreement.) The idea is to detect faults and then mask their results by always assigning a default value to their messages. In this way, the favored value of each fault-free processor becomes revealed to the other fault-free processors as the ability of the faulty processors to stop this from occurring is destroyed. Therefore, it is possible to switch between algorithms based on whether or not a persistent value, i.e., a value favored by a sufficient number of fault-free processors, exists. If one does not exist after a certain number of rounds then a large number of faulty processors have had to corrupt messages, and therefore have been detected by fault-free processors. If a persistent value does exist after a certain number of rounds, Bar-Noy et al. [1992] showed that it will not matter what the faulty processors do, as it is too late to stop the fault-free processors from agreeing on that value. The result is an algorithm that can tolerate $\lfloor (n - 1)/3 \rfloor$ faulty processors using $t + o(t)$ rounds of message passing, $O(n^b)$, $2 < b \le t$ message bits, and local computation time polynomial in n.

Similar techniques of hybridization of algorithms and utilization of fault detection to improve fault masking have been used by other researchers to produce efficient Byzantine agreement algorithms. Moses and Waarts [1988] reported an algorithm that requires $6t + 1$ processors, $O(t^8)$ message bits, and $t + 1$ rounds. Berman et al. [1989] reported a similar algorithm in terms of these cost measures, but which requires only $4t + 1$ processors.

Coan and Welch [1992] use hybridization to produce a more efficient algorithm than any of its constituent algorithms. In their approach, advantage is taken of the fact that fewer processors involved in Byzantine agreement results in fewer messages. Moreover, in order for a group of processors to report the results of a Byzantine agreement, only $2t + 1$ members need to broadcast the result such that the faulty processors cannot corrupt the transmission. Therefore, to reach agreement, the system is divided into a number of disjoint committees, each of which performs Byzantine agreement. Then the committees share their results via the broadcast described earlier. These steps are performed recursively to finally reach a global agreement. As a result, the hybrid algorithm uses $3t + 1$ processors, $t + o(t)$ rounds, and $O(t^2)$ message bits. A similar proposal for reducing the message costs involved in reaching Byzantine agreement is found in Barborak and Malek [1993].

5.2.3 Communication Redundancy for Efficient Agreement

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 [1985] looked at masking of these processors in the space domain. 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 buses, and each processor is given ports to these lines. Cristian [1989] showed that in these systems, t faults may be masked with no more that $t + 1$ messages on $b = t + 1$ buses regardless of the total number of processors. Moreover, some diagnostic information can be gathered from each broadcast.

5.2.4 Randomized Byzantine Agreement

Randomized Byzantine agreement algorithms have been proposed for their lower average number of rounds and lower 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 $t + 1$; yet Bracha [1987a] proved that randomized algorithms for asynchronous systems require the number of PEs still to be greater than or equal to $3t + 1$.

The idea of randomizing the agreement algorithm 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 situtations 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.

Randomized Byzantine agreement was first studied by Rabin [1983], Ben-Or [1983], and Bracha and Toueg [1983] primarily for asynchronous systems. Unfortunately, these algorithms require the system to be initialized with a list of coin flips created by some "trusted dealer." Shamir [1979] gave a method of keeping the values on the list secret until they were needed thus making it impossible for faulty processors to predict coin flips. Once initialized, the algorithms would complete in a constant expected number of rounds. Along these lines, Bracha [1987b] presented an $O(\log n)$ expected-rounds algorithm, which is better than the lower bound of $t + 1$ rounds for deterministic algorithms, for $t = n/(3 + \epsilon)$, $\epsilon > 0$.

Chor and Coan [1985] proposed an efficient algorithm for synchronous systems that uses $3t + 1$ processors, expected $O(t/\log n)$ rounds, and $O(n^2 t/\log n)$ messages that does not require an initialization of the system. 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 have the same value, where n is the number of processors and where t is the maximum number of those which may be faulty, 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 [1985] 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.

This algorithm given by Chor and Coan [1985] is considered too slow as the global coin flip algorithm must be performed every time it is needed. Feldman and Micali [1988] presented a technique to initialize the system with a large number of coin flips without the aid of a trusted dealer, in a constant expected number of rounds and in a way that no processor knows the value of the next flip until it is needed. Thus once the system is initialized, a number of "fast" agreements may take place.

Ben-Or [1983] and Bracha and Toueg [1985] studied randomized agreement in the case of fail-stop processors with the result that $n/2$ faults could be tolerated. Chor et al. [1989] examined randomized agreement for the fail-stop and omission fault models.

5.2.5 Approximate Agreement

In situations where each processor holds an estimate of some global value, such as the time, or an external variable measured by sensors connected to each processor, it is no longer necessary to reach exact agreement. If a group of sensors can measure a stimulus with a precision no greater than $\pm\epsilon$ then there is little point in having the processors agree on a more precise value. As a result, the problem of approximate agreement has been introduced.

Mahaney and Schneider [1985] studied the similar problem of inexact agreement. In this work it was assumed that all the processors' values were within some known bound prior to execution of the agreement algorithm.

Dolev et al. [1986] studied a protocol designed to reach a consensus on a real value rather than a discrete value. The rules for this approximate 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, although for the asynchronous environment $n \geq 5t + 1$.

Fekete [1991] also studied approximate-agreement algorithms and gave asymptotically optimal procedures. 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. Mahaney and Schneider [1985] used a similar approach, i.e., detecting failure to aid agreement.

5.2.6 Diagnosis of Byzantine Processors

Dolev [1982] noted that the bound on the number of faulty processors assumed for nonauthenticated Byzantine agreement could be used to diagnose those faulty processors. As a result, he introduced a new problem called *Crusader agreement* based on the following two requirements:

- $C1$ All the reliable processors that do not explicitly know that a processor z is faulty agree on the same value for z.
- $C2$ If a processor is fault free, then all the fault-free processors agree on its value.

Compared to the interactive consistency requirements for Byzantine agreement given in Section 2.2, condition *C2* is the same as the *consistency* condition. Condition *C1* is slightly different from *meaningfulness* in that only fault-free processors that are not certain z is faulty must agree on a value for z. This leads us to ask: how does a processor know when another processor is faulty?

Consider a point-to-point network whose connectivity is $2t + 1$ where t is the maximum number of faulty PEs, and $n \geq 3t + 1$ where n is the number of PEs in the system. Note that both of these conditions are necessary for reaching Byzantine agreement and, therefore, Crusader agreement, since Dolev [1982] showed that an algorithm that reaches Crusader agreement may be used to reach Byzantine agreement, i.e., a system capable of reaching Crusader agreement must also be capable of reaching Byzantine agreement. Now suppose that a processor u sends its value to each other processor in the system $2t + 1$ times along disjoint paths which will exist due to the connectivity of the system graph, according to the Menger theorem [Harary 1972]. A reliable processor v receives a set of messages, along with the route they supposedly traveled, transferred from u possibly through all fault-free processors and possibly through a number of faulty processors. As a result, v tries to find a set of no more than t processors which, when removed from the system along with all the messages they transferred, leave the remaining messages in agreement. If no such set of t or fewer processors exists, then v knows explicitly that u is faulty [Dolev 1982]. This does not mean that every fault-free processor knows that u is faulty. Based on this ability to diagnose faulty processors, Dolev gave an algorithm to reach Crusader agreement, which he claims requires many fewer messages than Byzantine agreement. A similar approach with the added benefit of directly testing each processor was examined by Yang and Masson [1988b] and is described here in Section 5.1.5.

Shin and Ramanathan [1987] used a similar approach as Dolev [1982] to diagnose systems suffering from Byzantine failures, except in this case, authentication is used. Once again, only the original transmitter of a message is diagnosed by a processor that receives that message, but in this situation, the information is collected over several rounds of agreement or perhaps over a mission in which repair or removal of processors is not feasible. At the time of diagnosis, the processors use authenticated Byzantine agreement to pass the syndrome information that each has collected so that each fault-free processor has the diagnostic information collected by each of the other fault-free processors. From this complete syndrome, each processor attempts to diagnose the faulty processors in the system which may or may not be possible. The authors show that in a system with $n > 2t + 2$ processors, a faulty processor will be correctly diagnosed if it is identified as faulty by at most $t + 1$ processors.

Walter [1990] described a similar approach to that of Shin and Ramanathan [1987] which performs diagnosis in the Multicomputer Architecture for Fault Tolerance (MAFT). In this case, though, diagnosis is performed on-line as each processor collects diagnostic information about the other processors in the system rather than after some mission time.

Adams and Ramarao [1989] abstracted the task of diagnosis to handle situations in which diagnosis is based on value discrepancies during Byzantine agreement, test results during system diagnosis, or any operation in which one processor gains information used to verify *evidence* that another processor is faulty. Their approach is to allow the processors to run for a certain amount of time, e.g., a testing round or a mission, collect these *evidences*, then distribute these evidences such that each processor may perform a diagnostic algorithm on them. The authors gave an optimal diagnostic algorithm in the respect that no other algorithm would be able to correctly diagnose more faulty processors from the evidences.

5.2.7 Related Problems to Byzantine Agreement

A number of problems have been presented that are similar in nature to Byzantine agreement and that possess similar requirements. In Fischer et al. [1986], weak Byzantine agreement, the Byzantine firing-squad problem, approximate agreement, and clock synchronization are examined. It was found that they all require that $n \geq 3t + 1$ must hold and that the connectivity of the communication graph must be at least $2t + 1$.

Weak Byzantine agreement is similar to Byzantine agreement in that one processor is trying to broadcast its value v to every other fault-free processor, but in this case, it is only necessary for the fault-free processors to agree on the value v if every processor in the system is fault free [Lamport 1983]. Otherwise, the fault-free PEs could agree on some default value. An instance of weak Byzantine agreement is the transaction commit problem for distributed databases. Interestingly, this problem is only solvable when Byzantine agreement is solvable, except when processors are allowed to send an infinite number of messages prior to making a decision (see Section 2.2).

The Byzantine firing squad problem is similar to the weak Byzantine general problem [Coan et al. 1985; Burns and Lynch 1987]. In this case, a stimulus is provided which ought to put all processors into a FIRE state at time T. The rules of the problem are (1) if all the processors are fault free and (2) at least one receives the stimulus, then after some finite delay, all will enter the FIRE state. If all processors are fault free and if no stimulus is received then no processors will enter the FIRE state. Finally, if one fault-free processor enters the FIRE state at time T then all fault-free processors enter the fire state at T. Once again, the problem has the same requirements as Byzantine agreement.

An agreement algorithm with decidedly fewer requirements than Byzantine agreement is the Part-Time Parliament problem or Paxon Synod protocol [Lamport 1989]. In this case, each processor must maintain a local copy of a distributed database despite the effects of any number of processors failing from fail-stop, crash, omission, or timing faults. Consistency of the local databases is maintained although the time to update a local database may be arbitrarily long; thus any local database may be out-of-date. Also, agreements to change the database may be made in an order different from the order of the initial requests for those agreements. Lamport showed that these limitations are sufficient to implement a state machine (as long as it is satisfactory for the machines to be unsynchronized for an arbitrary length of time) at a greatly reduced cost over implementation via Byzantine agreement.

A fundamental problem of fault-tolerant multiprocessor systems is the acceptance of data from an external source. If the broadcast mechanism for this source suffers 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 has arrived. This will make the system view of the input consistent. Krol [1991] introduced a set of algorithms called *Dispersed Joined Communication* (DJC) algorithms which solve the Input Problem more efficently, and in fact, which include the algorithm given by Pease et al. [1980] as a special case.

Classes $\mathscr{A}(t, k, s, \mathbf{D}, \mathbf{N})$ 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 [1991] 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 $\mathscr{A}(t, k, s, \mathbf{D}, \mathbf{N})$ is nonempty if and only if $|\mathbf{N}| \geq 2t + k$. He went on to give recursive procedures for designing these algorithms.

Krol [1991] also showed that the interactive consistency properties, which hold for all Byzantine agreement algorithms, hold for the class of algorithms $\mathscr{A}(t, k, s, \mathbf{D}, \mathbf{N})$ 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, unauthenticated Byzantine agreement algorithms.

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. Here we compare 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 and locate 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 diagnostic 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 the processor 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 mask faults, respectively, but can meet stricter deadlines.

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 unless the network is replicated (although, see Section 5.2.2). 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 time-outs or broadcasts. This is typically not examined, though, and would affect diagnosis efficiency.

Preparata et al. [1967] proved that for their model it is necessary that $n \geq 2t + 1$ for a system to be one-step t-diagnosable without repair. Depending on repairability, network structure, and the use of probabilistic approaches, though, this limit can change in either direction. Pease et al. [1980] proved that for Byzantine agreement, $n \geq 3t + 1$ is required to mask t malicious processors (also 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 [1987a; 1987b] showed that there is a nonzero probability of correct Byzantine agreement even when the number of faulty processors exceeds the resiliency bound. Somani et al. [1987] had similar results for t-diagnosable systems, and Vaidya and Pradhan [1991] showed that detection of more than t faults could be done inexpensively.

A qualitative comparison of system diagnosis and Byzantine agreement is elusive. That is, a comparison is difficult when there is a difference. Classically, Byzantine agreement is a problem of communication while system diagnosis is a problem of syndrome decoding. But due to similarities in purpose as well as a need to practically model similar systems, the areas are becoming more and more entwined. For example, research in distributed diagnosis, described in Section 5.1.5, has modeled faults very arbitrarily and therefore has resorted to Byzantine agreement algorithms to distribute diagnostic information. Research in Byzantine agreement has begun to look at how agreement information can be used to create syndromes that will allow a diagnosis of the system (see Section 5.2.6). As a result, there is very little if any difference left between certain branches of research on the Byzantine Generals Problem and on system diagnosis.

Powell [1992] looked at the effect of assumption coverage on system dependability. He attempted to quantify the advantages of a fault model that does not control the behavior of a faulty processor, i.e., a Byzantine fault model, over a fault model that demands a consistent behavior from a faulty processor, i.e., the PMC fault model, and over a fail-stop fault model. Of course the tradeoff is that $n \geq 3t + 1$ for Byzantine faults, $n \geq 2t + 1$ for the PMC model, and $n \geq t + 1$ for the fail-stop model. Thus, while Byzantine agreement covers a very weak fault assumption it is necessary to employ a great deal of redundancy, which means that there are more things to fail. Conversely, the fail-stop model may not be terribly realistic, but it requires a minimum amount of equipment; and therefore there are fewer things to fail. As a result, repair rates, failure rates, and fault bounds can affect which model will actually provide the most dependable system. In other words, Powell has shown that the assumption of Byzantine faults for critical applications is not always valid.

In conclusion of this section, Table 2 summarizes the models for consensus along with some major results for those models. Also, a relationship graph of the various consensus models presents (Figure 9) a visual overview of much of the work presented in this survey. In Figure 9, a node represents a model, and an edge between two model nodes represents a simple extension between those models. That is, a model will be related only to the most similar models by an edge labeled with the most significant difference between them.

7. APPLYING CONSENSUS PROTOCOLS

Consensus algorithms take many forms, and this section looks at some of the applications and implementations of this family of protocols. Preparata et al. [1967]

and Pease et al. [1980] 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.

7.1 Applications of Byzantine Agreement

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 [Schlichting and Schneider 1983]. Schneider [1984] used a consensus protocol to create a virtual fail-stop PE from a pool of processors that can exhibit Byzantine failures. 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.

Another popular application of consensus is the synchronization of a system of clocks. In fact, it was this synchronization problem that led to the development of interactive consistency algorithms for the SIFT project [Wensely et al. 1978]. Synchronization is a consensus on an arbitrary time value at some precise, real time. Lamport and Melliar-Smith [1984] used Byzantine agreement solutions to synchronize clocks in a system with faulty processors and clocks.

At Draper Labs, Byzantine agreement techniques were used to produce a Fault-Tolerant Processor (FTP) for aircraft, nuclear power plant, and submarine monitoring and control [Lala 1986; Lala et al. 1986]. In this project, a single, logical, fault-tolerant processor was implemented with four fault-intolerant processors that synchronized their work with Byzantine agreement protocols. This approach was taken to ensure that all faults would be masked at run time. In order to diagnose faults, tests were performed on the processors during idle periods. In the Fault-Tolerant Parallel Processor (FTPP) project (also at Draper Labs) work has been done to implement a parallel processor machine based on the concepts of Byzantine agreement [Harper et al. 1988]. In this case, special consideration was given to the communication network in order to alleviate the cost of $2t + 1$ connectivity on disjoint channels, which is necessary for Byzantine agreement [Dolev 1982].

The Philips Corporation has installed Byzantine agreement into their fault-tolerant switching system via the DJC algorithms given by Krol [1991] (see Section 5.2.2). In this (4, 2)-concept fault-tolerant computer [Krol 1982], there are four redundant processors accessing a single error-correcting coded memory divided into four separate modules. Krol [1991] 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.

7.2 Applications of Processor Membership

An early protocol designed to reach membership was implemented at Tandem Computers to manage the processors within their systems [Bartlett 1978]. 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 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. The protocol does not truly reach membership though as it is possible for p and q to have different views of the set of fault-free processors. For example, consider the case when p does not receive q's response to its "I'm alive!" message. Then suppose that q's response to p's second "I'm alive!" mes-

Table 2. Models for Consensus wıth Results

Model	Goal	Result	
Byzantine	Interactive	$n \geq 3t + 1$	[Pease et al. 1980]
Faults	Consistency	rounds $\geq t + 1$	[Fischer and Lynch 1982]
		connectivity $\geq 2t + 1$	[Dolev 1982]
		messages $\geq O(nt)$	[Dolev and Reischuk 1985]
		message bits $\geq O(t^2)$	[Dovel and Reischuck 1985]
		Synchronicity required	[Fischer et al. 1985]
		Algorithms—	[Coan and Welch 1992]
		$3t + 1$ processors	
		$t + O(t)$ rounds	
		$O(t^2)$ message bits	
	Approximate	Synchronıcity not required.	[Dolev et al 1986]
	Agreement	Algorithms	[Fekete 1991]
	Randomızed	$n \geq 3t + 1$	[Bracha 1987a]
	Agreement	rounds $< t + 1$	
Authenticated	Interactive	$n \geq t$	[Lamport et al. 1982]
Byzantıne	Consistency	sıgnatures $\geq O(nt)$	[Dolev and Reischuk 1985]
Faults		rounds $\geq t + 1$	[Dolev and Strong 1983]
		connectivity $\geq t + 1$	[Lamport et al. 1982]
		messages $\geq O(n + t^2)$	[Dolev and Reıschuk 1985]
		Algorithms—	[Dolev and Reıschuk 1985]
		$O(t)$ rounds	
		$O(n + t^2)$ messages	
Timing Faults	Membership	Algorithms—	[Cristıan 1991b]
		$n \geq t$	
PMC Model	t-diagnosis	$n \geq 2t + 1$	[Preparata et al. 1967]
	(one-step)	Characterization—	[Hakimi and Amin 1974]
		t tests per PE	
		$O(n^{2.5})$ diagnosis	[Dahbura and Masson 1984a]
		Diagnosis in arbitrary	[Fujıwara and Kinoshita 1978]
		systems is NP-complete.	
		Diagnosability—	[Sullıvan 1984]
		$O(\|E\|n^{1.5})$ algorithm	
	(sequential)	$n > 2t + 1$	[Preparata et al. 1967]
		Characterization	[Huang et al. 1989]
		Finding diagnosability	[Raghavan and Trıpathi 1991b]
		ıs co-NP-complete.	
	(set diagnosıs)	Finding t/s-dıagnosabilıty	[Sullivan 1984]
		ıs co-NP-complete.	
		Algorithms—	[Dahbura and Masson 1984a]
		$O(n^{2.5})$ diagnosis for $t = s$	

sage also fails. In this situation, q has received p's messages and therefore includes p in its membership, but p believes that q has failed and thus excludes q from its membership. Therefore, the Tandem scheme does not maintain a consistent view of the set of fault-free processors and cannot be called a membership protocol.

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 time-out allowing a "follower" processor to use checkpointing or simultaneous execution to take the role as leader. This method is similar to Cristian's [1991] membership protocols (also [Bartlett 1978]).

The Delta-4 system 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 with-

Table 2—*Continued*

Model	Goal	Result	
BGM Model	t-diagnosis (one-step)	Characterization— $n - 2 \geq t$	[Barsi et al. 1976]
		$O(nt^2)$ diagnosis	[Meyer 1983]
Comparison Testing	t-diagnosis (one-step)	Model	[Malek 1980] [Chwa and Hakimi 1981a]
		Model using PEs as comparators	[Maeng and Malek 1981]
		Characterization— Testing assignment connectivity $\geq t$	[Sengupta and Dahbura 1989] [Ammann and Dal Cın 1981]
		Diagnosis in arbitrary systems is NP-complete.	[Blough and Pelc 1992]
		Algorithms— $O(n^5)$ diagnosis for Maeng/Malek model.	[Sengupta and Dahbura 1989]
Adaptive Testing	t-diagnosis	tests $\geq n + t - 1$	[Blecher 1983]
		Algorithms— $O(\log_{\lfloor n/t \rfloor} t)$ rounds $O(n)$ tests	[Schmeicel et al. 1988]
Intermittent Faults	t_i-diagnosis	Characterization— $\lfloor 2t/3 \rfloor \leq t_i \leq t$ for a t-diagnosable testing assignment.	[Mallela and Masson 1978]
		$O(\lvert E \rvert)$ diagnosis	[Yang and Masson 1986]
Probabilistic Faults	p-t-diagnosis	Characterization	[Maheshwari and Hakimi 1976]
		$O(n^{2.5})$ diagnosis	[Dahbura and Masson 1984a]
		Finding diagnosability is NP-hard.	[Sullivan 1987]

out the fail-stop requirement [Barrett et al. 1990].

7.3 Applications of System-Level Diagnosis

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 as *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 [1985] suggested a broadcast network connecting a pool of processors and controlled by a reliable scheduler and diagnostics manager as a fault-tolerant architecture. Diagnosis and fault tolerance is achieved via 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 matches results.

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

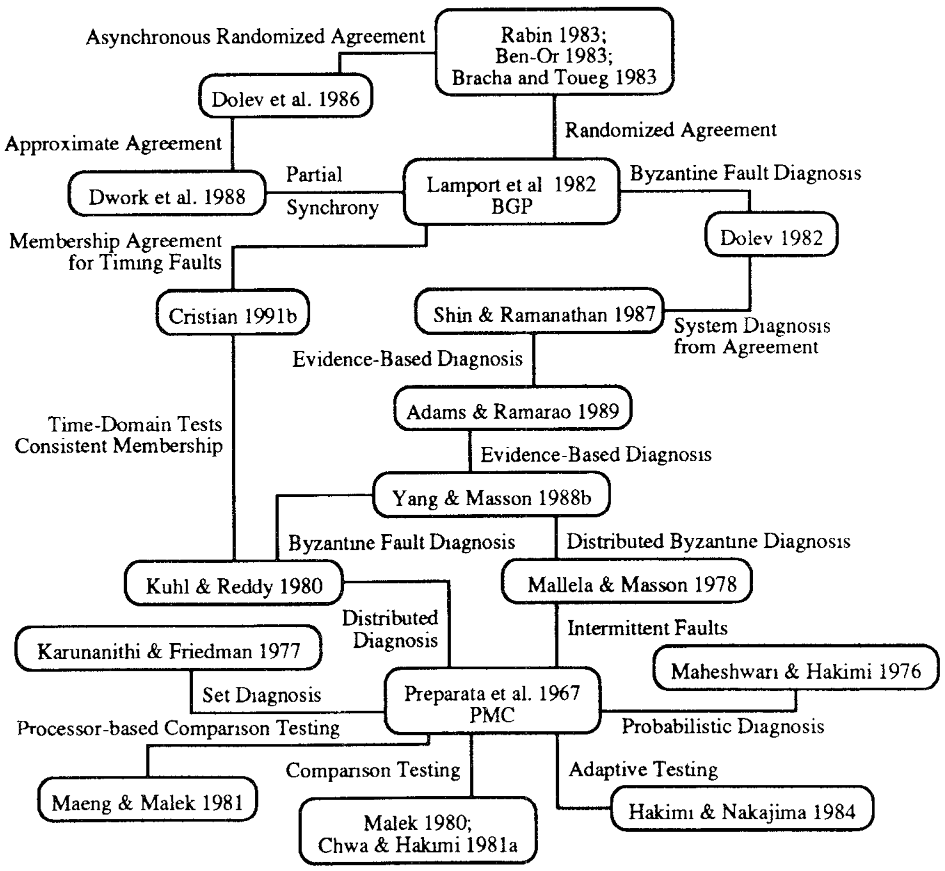


Figure 9. Consensus models

the intact configuration of the multiprocessor [Moritzen 1984]. Maehle reported that system diagnosis worked without problem over the five-year (1985–1990) lifespan of the DIRMU system (private communication, Oct. 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. [1984] [Bianchini et al. 1990]. The CMU diagnosis solution called **EVENT_SELF** enhances the **NEW_SELF** algorithm by (1) using time to synchronize tests and detect faulty PEs, by (2) allowing for PEs to join and leave the set of fault-free PEs, by (3) calling for reconfiguration testing after PE failures, and by (4) assuming system stability to minimize overhead. It does not test for faulty communication links.

EVENT_SELF 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 [Bianchini et al. 1990] devised an event-driven algorithm in which diagnostic information is not passed unless a fault or join occurs.

Table 3 compares major characteristics of **NEW_SELF** [Hosseini et al. 1984], Cristian's processor membership [Cristian 1991b], and **EVENT_SELF** [Bianchini et al. 1990].

Bianchini and Buskens [1991] continued the experiment at CMU with an adaptive distributed-system diagnosis algorithm (see Section 5.1.2) appropriately called **Adaptive DSD**. 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. That is, once a PE is diagnosed as fault free, the tests it performs are considered reliable, and therefore, any other PE should only be tested once by this fault-free PE to correctly determine its status. Thus, the testing assignment is adapted such that processors diagnosed as fault free perform all the testing in the system. By validating diagnosis information with a test of the source of the message, the algorithm tolerates a tester becoming faulty. As all processors must be tested by a fault-free processor to obtain a correct diagnosis, the minimum number of required tests is n. **Adaptive DSD** meets this lower bound while still guaranteeing correct diagnosis despite an unlimited number of faulty PEs. Furthermore, the message count is optimal at only $n\Delta t$, where Δt is the difference in the number of faults in the current round of testing and the number of faults in the previous round of testing. This is optimal as Δt represents the number of processor status changes, and for each change one message must be sent to each of n processors; thus, at least $n\Delta t$ messages are required. The cost of using a minimal number of tests and a minimal number of messages is reflected in the diagnosis latency which in the worst case is t times greater than that for **EVENT_SELF**.

Adaptive DSD assumes that the system is logically fully connected and that communication links are reliable. Stahl et al. [1992] relaxed these assumptions in their algorithm **Adapt** which works on-line in arbitrarily connected topologies even in the case of unreliable communication links. The algorithm maintains a minimally strong testing graph in each component of the system, i.e., if the system becomes disconnected, each component will continue to be diagnosed correctly. Testing is performed according to the testing graph which is in turn determined by the topology of the system. Therefore, the performance of the algorithm varies with the topology of the system on which it is executing. During periods in which no faults occur, the number of tests required may range from n to $2(n - 1)$, but in some "diabolical" cases this number may go as high as twice the number of edges in the system communication graph during a processor failure.

8. FUTURE RESEARCH

The trend in consensus problem research has been toward 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.

Table 3. A Comparison of Self-Diagnosis

Strategy	NEW_SELF	Processor Membership	EVENT_SELF
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 emmbedded 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.
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.

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 [1991b] and 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 diagnostic 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, as in Barborak and Malek [1992]. 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 demand for responsive (fault-tolerant, real-time) systems [Malek 1990], to construct diagnostic 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. [1985] showed the importance of synchronization and time and that ignorance of the time performance of a processor can render Byzantine agreement impossible.

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 nonrepairable "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 in 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 may occur so rarely as to make them probabilistically nonexistent. In this case, an algorithm with more stringent demands on the faulty hardware may perhaps be more efficient while achieving the same results.

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 to examine a generic framework which will apply to any system, e.g., computing, economics, government, etc., which is reminiscent of Byzantine agreement, or to 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 hope 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.

ACKNOWLEDGMENTS

The authors would like to thank Fred Schneider, Leslie Lamport, and the anonymous referees for their insightful comments.

REFERENCES

ADAMS, J., AND RAMARAO, K. 1989 Distributed diagnosis of Byzantine processors and links. In the *9th International IEEE Conference on Distributed Computing Systems*. IEEE, New York, 562–569.

AGRAWAL, P 1985. RAFT: A recursive algorithm for fault tolerance. In the *International Conference on Parallel Programming*, 814–821.

AMMANN, E., AND DAL CIN, M. 1981. Efficient algorithms for comparison-based self-diagnosis In *Self-Diagnosis and Fault-Tolerance*. Werkhefte der Universitat Tübingen, 4 Attempto-Verlag, Tubingen, 1–18.

ATTIYA, C , DOLEV, D., AND GIL, J. 1984. Asynchronous Byzantine consensus. In the *3rd Annual ACM Symposium on Principles of Distributed Computing*. ACM, New York, 119–133

BABAOĞLU, Ö. 1987a. Stopping times of distributed consensus protocols: A probabilistic approach. *Inf. Process. Lett. 25*, 3 (May), 163–169.

BABAOĞLU, Ö. 1987b. On the reliability of consensus-based fault-tolerant distributed computing systems. *ACM Trans. Comput. Syst.* 5, 3 (Nov.), 394–416.

BABAOĞLU, Ö., AND DRUMMOND, R. 1985. Streets of Byzantium: Network architectures for fast reliable broadcasts. *IEEE Trans. Softw. Eng. SE-11* (June), 546–554.

BAGCHI, A. 1992. A distributed algorithm for system-level diagnosis in hypercubes. In the *IEEE Workshop on Fault-Tolerant Parallel and Distributed Systems* (July). IEEE, New York, 106–113.

BAGCHI, A., AND HAKIMI, S. 1991. An optimal algorithm for distributed system level diagnosis. In the *21st International IEEE Symposium on Fault-Tolerant Computing*. IEEE, New York, 214–221.

BARBORAK, M., AND MALEK, M. 1993. Partitioning for efficient consensus. In *Proceedings of the 26th Hawaii International Conference on System Sciences* (Maui, Jan. 5–8), 438–446.

BAR-NOY, A., AND DOLEV, 1991. Consensus algorithms with one-bit messages. *Distrib. Comput. 4*, 105–110.

BAR-NOY, A., DOLEV, D., DWORK, C., AND STRONG, R. 1992. Shifting gears: Changing algorithms on the fly to expedite Byzantine agreement. *Inf. Comput. 97*, 205–233.

BARRETT, P., HILBORNE, A., BOND, P., SEATON, D., VERISSIMO, P., RODRIGUES, L., AND SPEIRS, N. 1990. The Delta-4 Extra Performance Architecture (XPA). In the *20th International IEEE Symposium on Fault-Tolerant Computing*. IEEE, New York, 481–488.

BARSI, F., GRANDONI, F., AND MAESTRINI, P. 1976. A theory of diagnosability of digital systems. *IEEE Trans. Comput. C-25*, 6 (June), 585–593.

BARTLETT, J. 1978. A 'non-stop' operating system. In *Proceedings of the Hawaii International Conference on System Sciences*, 103–119.

BEN-OR, M. 1983. Another advantage of free choice: Completely asynchronous agreement protocols. In the *2nd Annual ACM Symposium on Principles of Distributed Computing*. ACM, New York, 27–30.

BERMAN, P., AND PELC, A. 1990. Distributed probabilistic fault diagnosis for multiprocessor systems In the *20th International IEEE Symposium on Fault-Tolerant Computing*. IEEE, New York, 340–346.

BERMAN, P., GARAY, J., AND PERRY, K. 1989. Towards optimal distributed consensus. In the *30th IEEE Symposium on Foundations of Computer Science*. IEEE, New York, 410–415.

BERTSEKAS, D., AND GALLAGER, R. 1987. *Data Networks*. Prentice-Hall, Englewood Cliffs, N.J.

BIANCHINI, R., AND BUSKENS, R. 1991. An adaptive distributed system-level diagnosis algorithm and its implementation. In the *21st International IEEE Symposium on Fault-Tolerant Computing*. IEEE, New York, 222–229.

BIANCHINI, R., GOODWIN, K., AND NYDICK, D. 1990. Practical application and implementation of distributed system-level diagnosis theory. In the *20th International IEEE Symposium on Fault-Tolerant Computing*. IEEE, New York, 332–339.

BIRMAN, K. 1985. Replication and fault-tolerance in the ISIS system. In the *10th ACM SIGOPS Symposium on Operating Systems Principles*. (Dec.). ACM, New York, 79–86.

BIRMAN K., AND JOSEPH, T. 1987. Reliable communication in the presence of faults. *ACM Trans. Comput. Syst. 5*, 1 (Feb.), 47–76.

BLECHER, P. 1983. On a logical problem. *Discr. Math. 43*, 107–110.

BLOUGH, D., AND PELC, A. 1992. Complexity of fault diagnosis in comparison models. *IEEE Trans. Comput. 41*, 3 (Mar.), 318–323.

BLOUGH, D., AND PELC, A. 1990. Reliable diagnosis and repair in constant-degree multiprocessor systems. In the *20th International IEEE Symposium on Fault-Tolerant Computing*. IEEE, New York, 316–323.

BLOUGH, D., SULLIVAN, G., AND MASSON, G. 1992a. Intermittent fault diagnosis in multiprocessor systems. *IEEE Trans. Comput.* To be published.

BLOUGH, D., SULLIVAN, G., AND MASSON, G. 1992b. Efficient diagnosis of multiprocessor systems under probabilistic models. *IEEE Trans. Comput.* To be published.

BLOUGH, D., SULLIVAN, G., AND MASSON, G. 1989 Fault diagnosis for sparsely interconnected multiprocessor systems. In the *19th International IEEE Symposium of Fault-Tolerant Computing*. IEEE, New York, 62–69.

BLOUGH, D., SULLIVAN, G., AND MASSON, G. 1988. Almost certain diagnosis for intermittently faulty systems. In the *18th International IEEE Symposium of Fault-Tolerant Computing*. IEEE, New York, 260–265.

BLOUNT, M. 1978. Modeling of diagnosis in fail-softly computer systems. In the *8th International IEEE symposium on Fault-Tolerant Computing*. IEEE, New York, 53–58.

BLOUNT, M. 1977. Probabilistic treatment of diagnosis in digital systems. In the *7th International IEEE Symposium of Fault-Tolerant Computing*. IEEE, New York, 72–77.

BRACHA, G. 1987a. Asynchronous Byzantine agreement protocols. *Inf. Comput. 75* (Nov.), 130–143.

BRACHA, G. 1987b. An $O(\log n)$ expected rounds randomized Byzantine generals protocol. *J. ACM 34*, 4 (Oct.), 910–920.

BRACHA, G., AND TOUEG, S. 1985. Asynchronous consensus and broadcast protocols. *J. ACM 32*, 4 (Oct.), 824–840.

BRACHA, G., AND TOUEG, S. 1983. Resilient consensus protocols. In the *2nd ACM Symposium on Principles of Distributed Computing*. ACM, New York, 12–26.

BURNS, J., AND LYNCH, N. 1987. The Byzantine firing squad problem. *Adv. Comput. Res.: Parall. Distrib. Comput. 4*, 147–161.

CHOR, B., AND COAN, B. 1985. A simple and efficient randomized Byzantine agreement algorithm. *IEEE Trans. Softw. Eng. SE-11*, 6 (June), 531–539.

CHOR, B., MERRITT, M., AND SHMOYS, D. 1989. Simple constant-time consensus protocols in realistic failure models. *J. ACM 36*, 3 (July), 591–614.

CHWA, K., AND HAKIMI, S. 1981a. Schemes for fault-tolerant computing: A comparison of modularly redundant and *t*-diagnosable systems. *Inf. Contr. 49*, 212–238.

CHWA, K., AND HAKIMI, S. 1981b. On fault identification in diagnosable systems. *IEEE Trans. Comput. C-30*, 6 (June), 414–422.

CIOMPI, P., GRANDONI, F., AND SIMONCINI, L. 1981. Distributed diagnosis in multiprocessor systems: The MuTeam approach. In the *11th International IEEE Symposium on Fault-Tolerant Computing*. IEEE, New York, 25–29.

COAN, B. 1988. Efficient agreement using fault diagnosis. In *Proceedings of the 26th Allerton Conference on Communication, Control and Computing*. Univ. of Illinois, Urbana, Ill., 663–672.

COAN, B., AND WELCH, J. 1992. Modular construction of a Byzantine agreement protocol with optimal message bit complexity. *Inf. Comput. 97*, 61–85.

COAN, B., DOLEV, D., DWORK, C., AND STOCKMEYER, L. 1985. The distributed firing squad problem. In the *17th ACM Symposium on the Theory of Computing*. ACM, New York, 335–345.

CRISTIAN, F. 1991a. Understanding fault-tolerant distributed systems. *Commun. ACM 34*, 2 (Feb.).

CRISTIAN, F. 1991b. Reaching agreement on processor-group membership in synchronous distributed systems. *Distrib. Comput.* 4, 175–187.

CRISTIAN, F. 1990. Fault-tolerance in the advanced automation system. IBM Res. Rep. RJ 7424 (69595).

CRISTIAN, F. 1989. Synchronous atomic broadcast for redundant broadcast channels. IBM Res. Rep. RJ 7203.

CRISTIAN, F., AGHILI, H., STRONG, R., AND DOLEV, D. 1986. Atomic broadcast: From simple message diffusion to Byzantine agreement. IBM Tech. Rep. RJ 5244 (54244).

DAHBURA, A. 1988. System-level diagnosis: A perspective for the third decade. *AT&T Bell Laboratories Report. Concurrent Computations: Algorithms, Architecture, and Technology.* Plenum Press, New York.

DAHBURA, A. 1986. An efficient algorithm for identifying the most likely fault set in a probabilistically diagnosable system. *IEEE Trans. Comput. C-35*, 4 (Apr.), 354–356.

DAHBURA, A., AND MASSON, G. 1984a. An $O(n^{2.5})$ fault identification algorithm for diagnosable systems. *IEEE Trans. Comput. C-33*, 6 (June), 486–492.

DAHBURA, A., AND MASSON, G. 1984b. A practical variation of the $O(n^{2.5})$ fault diagnosis algorithm. In the *14th International IEEE Symposium on Fault-Tolerant Computing*. IEEE, New York, 428–433.

DAHBURA, A., AND MASSON, G. 1983a. Greedy diagnosis of hybrid fault situations. *IEEE Trans. Comput. C-32*, 8 (Aug.), 777–782.

DAHBURA, A., AND MASSON, G. 1983b. Greedy diagnosis of an intermittent-fault/transient-upset tolerant system design. *IEEE Trans. Comput. C-32*,10 (Oct.), 953–957.

DAHBURA, A., LAFERRERA, J., AND KING, L. 1985a. A performance study of system-level fault diagnosis algorithms (I). In the *4th International Conference on Computers and Communications*, 469–473.

DAHBURA, A., MASSON, G., AND YANG, C. 1985b. Self-implicating structures for diagnosable systems. *IEEE Trans. Comput. C-34*, 8 (Aug.), 718–723.

DAHUBURA, A., SABNANI, K., AND HERY, W. 1989. Spare capacity as a means of fault detection and diagnosis in multiprocessor systems. *IEEE Trans. Comput. C-38*, 6 (June), 881–891.

DAHBURA, A., SABNANI, K., AND KING, L. 1987. The comparison approach to multiprocessor fault diagnosis. *IEEE Trans. Comput. C-36*, 3 (Mar.), 373–378.

DAL CIN, M. 1984. Distributed diagnosis for computing networks. *Microprocess. Microprogram. 14*, 139–144.

DAL CIN, M. 1982. A diagnostic device for large multiprocessor systems. In the *12th International IEEE Symposium on Fault-Tolerant Computing*. IEEE, New York, 357–360.

DAL CIN, M. 1980. Self-testing and self-diagnosing multicomponent systems. In the *10th International IEEE Symposium on Fault-Tolerant Computing*. IEEE, New York.

DAL CIN, M. 1978. Performance evaluation of self-diagnosing multiprocessing systems. In the *8th International IEEE Symposium on Fault-Tolerant Computing*. IEEE, New York, 59–64.

DAL CIN, M., AND DILGER, E. 1981. On the diagnosability of self-testing multi-microprocessor systems. *Microprocess. Microprogram. 7*, 177–184.

DAL CIN, M., AND FLORIAN, F. 1985. Analysis of a fault-tolerant distributed diagnosis algorithm. In the *15th International IEEE Symposium on*

Fault-Tolerant Computing. IEEE, New York, 159–164.

DEGONIA, P., WITT, R., LAMPE, D., AND COLE, E. 1978. Micronet—A self-healing network for signal processing. In *the Government Microcircuit Application Conference*. (Nov.), 370–375.

DEO, N. 1974. *Graph Theory with Applications to Engineering and Computer Science*. Prentice-Hall, Englewood Cliffs, N.J.

DOLEV, D. 1982. The Byzantine generals strike again. *J. Alg. 3*, 14–30.

DOLEV, D. 1981. Unanimity in an unknown and unreliable environment. In the *22nd Symposium on the Foundations of Computer Science*, 159–168.

DOLEV, D., AND REISCHUK, R. 1985. Bounds on information exchange for Byzantine agreement. *J. ACM 32*, 1 (Jan.), 191–204.

DOLEV, D., AND STRONG, H. 1982. Authenticated algorithms for Byzantine agreement. *J. Alg.* 3, 14–30.

DOLEV, D., DWORK, C., AND STOCKMEYER, L. 1987. On the minimal synchronism needed for distributed consensus. *J. ACM 34*, 1 (Jan.), 77–97.

DOLEV, D., LYNCH, N., PINTER, S., STARK, E., AND WEIHL, W. 1986. Reaching approximate agreement in the presence of faults. *J. ACM 33*, 3 (July), 499–516.

DWORK, C., LYNCH, N., AND STOCKMEYER, L. 1988. Consensus in the presence of partial synchrony. *J. ACM 35*, 2 (Apr.), 288–323.

FEKETE, A. 1991. Asymptotically optimal algorithms for approximate agreement. *Distrib. Comput. 4*, 9–29.

FELDMAN, P., AND MICALI, S. 1988. Optimal algorithms for Byzantine agreement. In the *20th ACM Symposium on Theory of Computing* ACM, New York, 148–161.

FISCHER, M., AND LYNCH, N. 1982. A lower bound for the time to assure interactive consistency. *Inf. Process. Lett. 14*, 4 (June), 183–186.

FISCHER, M., LYNCH, N., AND MERRITT, M. 1986. Easy impossibility proofs for distributed consensus problems. *Distrib. Comput. 1*, 1, 26–39.

FISCHER, M., LYNCH, N., AND PATERSON, M. 1985. Impossibility of distributed consensus with one faulty process. *J. ACM 32*, 2 (Apr.), 374–382.

FRIEDMAN, A. 1975. A new measure of digital system diagnosis. In the *5th International IEEE Symposium on Fault-Tolerant Computing*. IEEE, New York, 167–169.

FRIEDMAN, A., AND SIMONCINI, L. 1980. System-level fault diagnosis. *Computer* (Mar.), 47–53.

FUJIWARA, H., AND KINOSHITA, K. 1978. On the computational complexity of system diagnosis. *IEEE Trans. Comput. C-27*, 10 (Oct.), 881–885.

FUJIWARA, H., AND OZAKI, H. 1979. On the diagnosability of systems with self testing units. In the *9th International IEEE Symposium on Fault-Tolerant Computing*. IEEE, New York, 157–160

FUSSELL, D , AND RANGARAJAN, S. 1989. Probabilistic diagnosis of multiprocessor systems with arbitrary connectivity. In the *19th International IEEE Symposium on Fault-Tolerant Computing*. IEEE, New York, 560–565.

GUPTA, R., AND RAMAKRISHNAN, I. 1987. System-level fault diagnosis in malicious environments. In the *17th International IEEE Symposium on Fault-Tolerant Computing*. IEEE, New York, 184–189.

HAKIMI, S., AND AMIN, A. 1974. Characterization of connection assignment of diagnosable systems. *IEEE Trans. Comput. C-23*, 1 (Jan.), 86–88.

HAKIMI, S., AND NAKAJIMA, K. 1984. On adaptive system diagnosis. *IEEE Trans. Comput. C-33*, 3 (Mar.), 234–240.

HARARY, F. 1972. *Graph Theory*. Addison-Wesley, Reading, Mass.

HARPER, R., LALA, J., AND DEYST, J. 1988. Fault tolerant parallel processor architecture overview. In the *18th International IEEE Symposium on Fault-Tolerant Computing*. IEEE, New York, 252–257.

HOLT, C., AND SMITH, J. 1981. Diagnosis of systems with asymmetric invalidation. *IEEE Trans. Comput. C-30*, 9 (Sept.), 679–690.

HOSSEINI, S., KUHL, J., AND REDDY, S. 1985. On self fault-diagnosis of the distributed systems. In the *15th International IEEE Symposium on Fault-Tolerant Computing*. IEEE, New York, 30–35.

HOSSEINI, S., KUHL, J., AND REDDY, S. 1984. A diagnosis algorithm for distributed computing systems with dynamic failure and repair. *IEEE Trans. Comput. C-33*, 3 (Mar.), 223–233.

HUANG, S., XU, J., AND CHEN, T. 1989. Characterization and design of sequentially t-diagnosable systems. In the *19th International IEEE Symposium on Fault-Tolerant Computing*. IEEE, New York, 554–559.

KAMEDA, T., TOIDA, S., AND ALLAN, F. 1975. A diagnosing algorithm for networks. *Inf. Contr. 29*, 141–148.

KARUNANITHI, S., AND FRIEDMAN, A. 1977. System diagnosis with t/s diagnosability. In the *7th International IEEE Symposium on Fault-Tolerant Computing*. IEEE, New York, 65–71.

KAVIANPOUR, A., AND FRIEDMAN, A. 1978. Efficient design of easily diagnosable systems. In *Proceedings of the 3rd USA-Japan Computer Conference*, 251–257.

KIM, K., AND YANG, S 1986. Fault tolerance mechanisms in real-time distributed operating systems: An overview. In the *Pacific Computer Communications '85*. Elsevier Science Publishers, 239–248.

KIME, C. 1970. An analysis model for digital system diagnosis *IEEE Trans. Comput. C-19*, 11 (Nov.), 1063–1073.

KIME, C. 1986. System diagnosis. In *Fault Tolerant Computing: Theory and Techniques*. Prentice-Hall, Englewood Cliffs, N.J.

KOHDA, T., AND ABIRU, K. 1988. A recursive procedure for optimally designing a hybrid fault diagnosable system. In the *18th International IEEE Symposium on Fault-Tolerant Computing* IEEE, New York, 272–277.

KOZLOWSKI, W., AND KRAWCZYK, H. 1991. A comparison-based approach to multicomputer system diagnosis in hybrid fault situations. *IEEE Trans. Comput. 40*, 11 (Nov.), 1283–1287

KREUTZER, S., AND HAKIMI, S. 1988. Distributed diagnosis and the system user. *IEEE Trans. Comput. 37*, 1 (Jan.), 71–78.

KREUTZER, S., AND HAKIMI, S. 1987. System-level fault diagnosis: A survey. *Microprocess. Microprogram. 20*, 323–330.

KREUTZER, S., AND HAKIMI, S. 1983. Adaptive fault identification in two new diagnostic models. In *Proceedings of the 21st Allerton Conference on Communication, Control and Computing* Univ. of Illinois, Urbana, Ill., 353–362.

KROL, T. 1991 A generalization of fault-tolerance based on masking. Ph.D. dissertation, Eindhoven Univ. of Technology, Eindhoven, The Netherlands.

KROL, T. 1982. The '(4, 2)-Concept' fault tolerant computer In the *12th International IEEE Symposium on Fault-Tolerant Computing*. IEEE, New York. 49–54.

KUHL, J., AND REDDY, S. 1981. Fault-diagnosis in fully distributed systems. In the *11th International IEEE Symposium on Fault-Tolerant Computing*. IEEE, New York, 100–105

KUHL, J., AND REDDY, S. 1980a Distributed fault-tolerance for large multiprocessor systems. In *Proceedings of the 7th Annual Symposium on Computer Architecture*, 23–30.

KUHL, J., AND REDDY, S. 1980b. Some extensions to the theory of system level fault diagnosis. In the *10th International IEEE Symposium on Fault-Tolerant Computing*. IEEE, New York, 291–296.

LALA, J. 1986 A Byzantine resilient fault tolerant computer for nuclear power plant applications. In the *16th International IEEE Symposium on Fault-Tolerant Computing*. IEEE, New York, 338–343.

LALA, J., ALGER, L., GAUTHIER, R., AND DZWONCZYK, M. 1986. A fault tolerant processor to meet rigorous failure requirements. In *Proceedings of the 7th AIAA-IEEE Digital Avionics Systems Conference*. AIAA-IEEE, New York, 555–562

LAMPORT, L. 1989. The part-time parliament *Digital Tech. Rep. 49* (Sept. 1).

LAMPORT, L. 1983. Weak Byzantine generals problem *J. ACM 30*, 3 (July), 668–676

LAMPORT, L., AND MELLIAR-SMITH, P. 1984. Byzantine clock synchronization. In the *3rd ACM Symposium on Principles of Distributed Computing*. ACM, New York, 68–74.

LAMPORT, L., SHOSTAK, R., AND PEASE, M. 1982. The Byzantine generals problem. *ACM Trans. Program. Lang. Syst. 4*, 3 (July), 382–401.

LARANJEIRA, L., MALEK, M., AND JENEVEIN, R. 1991. On tolerating faults in naturally redundant algorithms. In the *10th Symposium on Reliable Distributed Systems* (Pisa, Italy, Sept.). IEEE Computer Society, Los Alamitos, Calif., 118–127.

LEE, S., AND SHIN, K 1990. Optimal multiple syndrome probabilistic diagnosis. In the *20th International IEEE Symposium on Fault-Tolerant Computing*. IEEE, New York, 324–331.

LIAW, C., SU, S , AND MALAIYA, Y. 1982. Self-diagnosis of non-homogeneous distributed systems. In the *12th International IEEE Symposium on Fault-Tolerant Computing*. IEEE, New York, 349–352.

LOMBARDI, F 1985. Diagnosable systems for fault tolerant computing In the *15th International IEEE Symposium on Fault-Tolerant Computing*. IEEE, New York, 42–47.

MAEHLE, E., MORITZEN, K. AND WIRL, K. 1986. A graph model for diagnosis and reconfiguration and its application to a fault-tolerant multiprocessor system. In the *16th International IEEE Symposium on Fault-Tolerant Computing*. (Vienna, Austria). IEEE, New York, 292–297.

MAENG, J., AND MALEK, M 1981. A comparison connection assignment for self-diagnosis of multiprocessor systems. In the *11th International IEEE Symposium on Fault-Tolerant Computing*. IEEE, New York, 173–175.

MAHANEY, S., AND SCHNEIDER, F 1985. Inexact agreement· Accuracy, precision and graceful degradation In the *4th ACM Symposium on the Principles of Distributed Computing*. ACM, New York, 237–249.

MAHESHWARI, S., AND HAKIMI, S 1976. On models for diagnosable systems and probabilistic fault diagnosis. *IEEE Trans. Comput. C-25*, 3 (Mar.), 228–236.

MALEK, M. 1991. Responsive systems: A marriage between real-time and fault tolerance. In the *5th GI / NTG / GMR Conference on Fault-Tolerant Computing Systems*. Informatik-Fachberichte, vol. 283. Springer-Verlag, Berlin, 1–17.

MALEK, M. 1990. Responsive systems: A challenge for the nineties. In *Proceedings of Euromicro '90, 16th Symposium on Microprocessing and Microprogramming*. Microprocessing and Microprogramming 30, North-Holland, Amsterdam, 9–16.

MALEK, M. 1980. A comparison connection assignment for diagnosis of multiprocessor systems. In *Proceedings of the 7th Annual Symposium on Computer Architecture*, 31–36.

MALEK, M., AND LIU, K. 1980. Graph theory models in fault diagnosis and fault tolerance. In *Design Automation and Fault-Tolerant Computing*, vol. 3, 3, 4. Computer Science Press, 155–169.

MALLELA, S. 1980. On diagnosable systems with simple algorithms. In *Proceedings of the 1980 Conference on Information Science and Systems*. Princeton Univ., Princeton, N.J., 545–549.

MALLELA, S., AND MASSON, G. 1980. Diagnosis without repair for hybrid fault situations. *IEEE Trans. Comput. C-29*, 6 (June), 461–470.

MALLELA, S., AND MASSON, G. 1978. Diagnosable systems for intermittent faults. *IEEE Trans. Comput. C-27*, 6 (June), 560–566.

MAXEMCHUK, N., AND DAHBURA, A. 1986. Optimal diagnosable system design using full-difference triangles. *IEEE Trans. Comput. C-35*, 9 (Sept), 837–839.

MENEZES, B., JOHNSON, A., MALEK, M., JENEVEIN, R., AND YAU, K. 1992. Fault impact and fault tolerance in multiprocessor interconnection networks. In *Quality and Reliability Engineering International*, vol. 8, 485–500.

MEYER, G. 1983. A diagnosis algorithm for the BGM system-level fault model. In *Proceedings of the 21st Allerton Conference on Communication, Control and Computing*. Univ. of Illinois, Urbana, Ill., 345–351.

MEYER, G., AND MASSON, G. 1978. An efficient fault diagnosis algorithm for symmetric multiple processor architectures. *IEEE Trans. Comput. C-27*, 11 (Nov.), 1059–1063.

MORITZEN, K. 1984. System level fault-diagnosis in distributed systems. In the *2nd GI / NTG / GMR Conference on Fault-Tolerant Computing Systems*. Informatik-Fachberichte, vol. 84. Springer-Verlag, Berlin, 301–312.

MOSES, Y., AND WAARTS, O. 1988. $(t + 1)$-round Byzantine agreement in polynomial time. In the *29th Symposium on Foundations of Computer Science*, 246–255.

NAIR, R. 1978. Diagnosis, self-diagnosis and roving diagnosis. Dept. of Computer Science Rep. R-823, University of Illinois, Urbana, Ill.

NAKAJIMA, K. 1981. A new approach to system diagnosis. In *Proceedings of the 19th Allerton Conference on Communication, Control and Computing*. Univ. of Illinois, Urbana, Ill., 697–706.

PEASE, M., SHOSTAK, R., AND LAMPORT, L., 1980, Reaching agreement in the presence of faults. *J. ACM, 27*, 2 (Apr.), 228–234.

PELC, A. 1992. Optimal fault diagnosis in comparison models. *IEEE Trans. Comput. 41*, 6 (June), 779–786.

PELC, A. 1991. Undirected graph models for system-level fault diagnosis. *IEEE Trans. Comput. 40* 11 (Nov.), 1271–1276.

POWELL, D. 1992. Fault mode assumptions and assumption coverage. In the *22nd IEEE International Symposium on Fault-Tolerant Computing*. IEEE, New York, 386–395.

PRADHAN, D., AND REDDY, S. 1982. A fault-tolerant communication architecture for distributed systems. *IEEE Trans. Comput. C-31*, 9 (Sept.), 863–870.

PREPARATA, F., METZE, G., AND CHIEN, R. 1967. On the connection assignment problem of diagnosable systems. *IEEE Trans. Elect. Comput. EC-16*, 6 (Dec.), 848–854.

RABIN, M. 1983. Randomized Byzantine generals. In *Proceedings of the 24th Symposium on Foundations of Computer Science*, 403–409.

RAGHAVAN, V., AND TRIPATHI, A. 1991a. Improved diagnosability algorithms. *IEEE Trans Comput. 49*, 2 (Feb.), 143–153.

RAGHAVAN, V., AND TRIPATHI, A. 1991b. Sequential diagnosability is co-NP complete. *IEEE Trans. Comput. 40*, 5 (May), 584–595.

RANGARAJAN, S., AND FUSSELL, D 1991. Probabilistic diagnosis algorithms tailored to system topology. In the *21st International IEEE Symposium on Fault-Tolerant Computing*. IEEE, New York, 230–237.

RANGARAJAN, S. AND FUSSELL, D. 1988. A probabilistic method for fault diagnosis of multiprocessor systems. In the *18th International IEEE Symposium on Fault-Tolerant Computing*. IEEE, New York, 278–283.

RANGARAJAN, S., FUSSELL, D., AND MALEK, M. 1990. Built-in testing of integrated circuit wafers. *IEEE Trans. Comput. 39*, 2 (Feb.), 195–204.

RAYNAL, M. 1988. *Distributed Algorithms and Protocols*. John Wiley, New York, 137–163.

RUSSELL, J., AND KIME, C. 1975a. System fault diagnosis: Masking, exposure, and diagnosability without repair. *IEEE Trans. Comput. C-24*, 12 (Dec.), 1115–1161.

RUSSELL, J., AND KIME, C. 1975b. System fault diagnosis: Closure and diagnosability with repair. *IEEE Trans. Comput. C-24*, 11 (Nov.), 1078–1088.

SAHEBAN, F., SIMONCINI, L., AND FRIEDMAN, A. 1979. Concurrent computation and diagnosis in multiprocessor systems. In the *9th International IEEE Symposium on Fault-Tolerant Computing*. IEEE, New York, 149–156.

SCHEINERMAN, E. 1987. Almost sure fault tolerance in random graphs. *SIAM J. Comput. 16*, 6 (Dec.), 1124–1134.

SCHLICHTING, R., AND SCHNEIDER, F. 1983. Fail-stop processors: An approach to designing fault-tolerant computing systems. *ACM Trans. Comput. Sys. 1*, 3 (Aug.), 222–238.

SCHMEICHEL, E., HAKIMI, S., OTSUKA, M., AND SULLIVAN, G. 1988. On minimizing testing rounds for fault identification. In the *18th International IEEE Symposium on Fault-Tolerant Computing*. IEEE, New York, 266–271.

SCHNEIDER, F. 1990. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Comput. Surv. 22*, 4 (Dec.), 299–319.

SCHNEIDER, F. 1984. Byzantine generals in action: Implementing fail-stop processors. *ACM Trans. Comput. Syst. 2*, 2 (May), 145–154.

SENGUPTA, A., AND DAHBURA, A. 1989 On self-diagnosable multiprocessor systems. Diagnosis by the comparison approach. In the *19th International IEEE Symposium on Fault-Tolerant Computing*. IEEE, New York, 54–61.

SHAMIR, A. 1979. How to share a secret. *Commun. ACM 22*, 11 (Nov.), 612–613.

SHIN, K., AND RAMANATHAN, P 1987. Diagnosis of processors with Byzantine faults. In the *17th International IEEE Symposium on Fault-Tolerant Computing*. IEEE, New York, 55–60

SIMONS, B., AND SPECTOR, A., EDS. 1990. *Lecture Notes in Computer Science: Fault-Tolerant Distributed Computing*, vol. 448. Springer-Verlag, Berlin.

SMITH, J. 1979. Universal system diagnosis algorithms. *IEEE Trans. Comput. C-28*, 5 (May), 374–378.

SOMANI, A., AND AGARWAL, V 1992. Distributed diagnosis algorithms for regular interconnected structures. *IEEE Trans. Comput. 41*, 7 (July), 899–906.

SOMANI, A., AGARWAL, V., AND AVIS, D. 1987. A generalized theory for system level diagnosis *IEEE Trans. Comput. C-36*, 5 (May), 538–546.

STAHL, M., BUSKENS, R., AND BIANCHINI, R. 1992. On-line diagnosis in general topology networks. In the *IEEE Workshop on Fault-Tolerant Parallel and Distributed Systems*. IEEE, New York, 114–121.

SULLIVAN, G. 1988. An $O(t^3 + |E|)$ fault identification algorithm for diagnosable systems. *IEEE Trans. Comput. 37*, 4 (Apr), 388–397.

SULLIVAN, G 1987. System-level fault diagnosability in probabilistic and weighted models. In the *17th International IEEE Symposium of Fault-Tolerant Computing*. IEEE, New York, 190–195.

SULLIVAN, G. 1984. A polynomial time algorithm for fault diagnosability. In the *25th Symposium on the Foundations of Computer Science*, 148–156.

TOUEG, S., PERRY, K., AND SRIKANTH, T. 1987. Fast distributed agreement. *SIAM J Comput. 16*, 445–458.

TUREK, J., AND SHASHA, D. 1992 The many faces of consensus in distributed systems. *Computer 18* 6 (June), 8–17.

TURPIN, R., AND COAN, B. 1984. Extending binary Byzantine agreement to multivalued Byzantine agreement. *Inf. Process. Lett. 18* (Feb.), 73–76.

VAIDYA, N., AND PRADHAN, D. 1991. System level diagnosis: Combining detection and location In the *21st International IEEE Symposium on Fault-Tolerant Computing*, IEEE, New York, 488–495.

WALTER, C. 1990. Identifying the cause of detected errors. In the *20th International IEEE Symposium on Fault-Tolerant Computing*. IEEE, New York, 48–55.

WENSLEY, J , LAMPORT, L., GOLDBERG, J., GREEN, M , LEVITT, K., MELLIAR-SMITH, P., SHOSTAK, R , AND WEINSTOCK, C. 1978. SIFT: Design and analysis of a fault-tolerant computer for aircraft control. *Proc IEEE, 66*, 10 (Oct.), 1240–1255.

XU, J. 1991. The $t/(n-1)$-diagnosability and its applications to fault tolerance. In the *21st International IEEE Symposium on Fault-Tolerant Computing*. IEEE, New York, 496–503.

YANG, C., AND MASSON, G. 1988a. Hybrid fault diagnosability with unreliable communication links. *IEEE Trans. Comput. 37*, 2 (Feb), 175–181.

YANG, C., AND MASSON, G 1988b. A distributed algorithm for fault diagnosis in systems with soft failures *IEEE Trans. Comput. 37*, 11 (Nov.), 1476–1480.

YANG, C., AND MASSON, G. 1987. A new measure for hybrid fault diagnosability. *IEEE Trans Comput. C-36*, 3 (Mar.), 378–383.

YANG, C., AND MASSON, G 1986. An efficient algorithm for multiprocessor fault diagnosis using the comparison approach. In the *16th International IEEE Symposium on Fault-Tolerant Computing* (Vienna, Austria). IEEE, New York, 238–243.

YANG, C., AND MASSON, G. 1985a. A fault identification algorithm for t_i-diagnosable systems. In the *15th International IEEE Symposium on Fault-Tolerant Computing*. IEEE, New York, 78–83.

YANG, C., AND MASSON, G. 1985b. A generalization of hybrid faulty diagnosabily. In the *15th International IEEE Symposium on Fault-Tolerant Computing*. IEEE, New York, 36–41.

YANG, C., MASSON, G., AND LEONETTI, R. 1986. On fault isolation and identification in t_1/t_1-diagnosable systems. *IEEE Trans. Comput. C-35*, 7 (July), 639–643.

Received November 1991; final revision accepted December 1992.