Formally Verified On-Line Diagnosis

Chris J. Walter	Patrick Lincoln	Neeraj Suri
WW Technology Group	SRI International	Dept. of CIS, NJIT
Ellicott City, MD	Menlo Park, CA	Newark, NJ
cwalter@blaze.jhu.edu	lincoln@csl.sri.com	suri@cis.njit.edu

Abstract

A reconfigurable fault tolerant system achieves the attributes of dependability of operations through fault detection, fault isolation and reconfiguration, typically referred to as the FDIR paradigm. Fault diagnosis is a key component of this approach, requiring an accurate determination of the health and state of the system. An imprecise state assessment can lead to catastrophic failure due to an optimistic diagnosis, or conversely, result in underutilization of resources because of a pessimistic diagnosis. Differing from classical testing and other off-line diagnostic approaches, we develop procedures for maximal utilization of the system state information to provide for continual, on-line diagnosis and reconfiguration capabilities as an integral part of the system operations. Our diagnosis approach, unlike existing techniques, does not require administered testing to gather syndrome information but is based on monitoring the system message traffic among redundant system functions. We present comprehensive on-line diagnosis algorithms capable of handling a continuum of faults of varying severity at the node and link level. Not only are the proposed algorithms on-line in nature, but are themselves tolerant to faults in the diagnostic process. Formal analysis is presented for all proposed algorithms. These proofs offer both insight into the algorithm operations and facilitate a rigorous formal verification of the developed algorithms.

Keywords: Fault Diagnosis, On-Line, Fault Handling, Formal Methods

1 Introduction

The goal of an effective fault tolerant system is to provide maximal sustained system dependability at the lowest possible cost to the user. The ability to accurately diagnose and recover from faults constitutes an important element in achieving this goal. In general, fault diagnosis has two primary objectives: (a) to identify a faulty unit so as to restrict the effect of faults on the system operations, and (b) to support the fault isolation and reconfiguration process with information concerning the state of the system resources. The better the diagnostic resolution for identifying a fault and the corresponding system state, the more accurate is the FDIR procedure, with minimal and highly localized resources targeted for repair. This strategy results in efficient management of the system resources, and the enhanced diagnostic resolution can help achieve lower repair costs (time latency of fault recovery, cost of spares and fault handling, etc.). Also, if diagnosis and fault handling can be done quickly there will be less exposure time to faults, reducing the likelihood of fault accumulation

that could eventually overwhelm the system's fault tolerance mechanisms. Ultimately, when FDIR is performed in conjunction with ongoing system operations, it translates to having more resources available to sustain system operations in the presence of faults, thus prolonging the availability and reliability of the system.

A variety of approaches have been proposed for system diagnosis — an excellent survey appears in Barborak, Malek and Dahbura[1]. Most classical approaches vary on the theme of the original Preparata, Metze and Chen (PMC) model[21]; requiring processors to conduct tests on other processors, and subsequently interpreting the accumulated test results to determine the status of a processor. However, for response critical systems and for covering faults of unrestricted severity, the previously proposed methods have limited applicability. First, these methods are invariably off-line techniques due to the extensive overhead incurred in collecting and interpreting the diagnostic information. Second, each approach is based on a particular system and fault model (data-domain or time-domain) which influences the amount of diagnostic resolution obtainable and determines the achievable level of system dependability. Third, diagnosis is treated as an isolated objective in the system and not as a continual and integral phase of the system operations.

Thus, our interest lies in considering two fundamental issues in fault diagnosis in distributed systems, namely (a) developing new diagnostic approaches for covering sets[1] of faults of varying severity including the extremal (and pessimistic) cases of arbitrary (Byzantine) faults, and (b) to develop effective and formally verified on-line fault diagnosis and recovery strategies. For a better perspective on these issues we provide a brief commentary on each.

Considering the issue of the nature of faults, a seminal paper in the area of diagnosis is the PMC model[21] which studied diagnosability in distributed systems using explicitly administered tests. The PMC approach and its variants consider a class of systems which can be partitioned into units that are capable of testing each other. Unit-to-unit testing assignments are established to obtain test syndromes, followed by an exchange of syndrome information across all units to get a global picture of the system status. It is assumed that there is a bound on the number of potentially faulty units and that the test syndromes obtained from a faulty unit are unreliable. Initial development of the PMC model focused on permanent faults so that repeatable results could be relied upon when any fault-free unit tested a faulty unit. Derivatives and subsequent work extended the fault model to cover transient, intermittent and link faults. However, these approaches assume failures only for a specified and distinct fault class. A very small number handle a combination of hard and soft failures but still rely on administered testing[39]. For a detailed discussion and completeness of references, we refer the reader to the survey article of Barborak, Malek and Dahbura[1]

However, our interest lies in developing algorithms which are capable of handling a wide range of faults – from easily distinguishable fail-stop or other easily detectable faults to those of an unrestricted nature – all within a common algorithmic framework, and on an on-line basis. Our intent is to consider real system environments where faults come as progression or combination of various types (transients, intermittent, arbitrary faults etc.) and not as convenient (and unrealistic) single fault instances. We are interested in covering, both, a wider range of fault types and instances of combinations of fault types appearing together.

The challenges and benefits for achieving high resolution diagnosis can be contrasted for both the off-line and on-line environments. Each circumstance aims to accurately assess the state of

[1] instead of the limitation of considering a discrete fault class in existing approaches.

the system resources. Off-line diagnosis procedures can only support identification of faulty units needing repair, they cannot support reconfigurable fault tolerance since the results are not available during active operation. A second problem is that transient or intermittent faults are not easily reproducible in an off-line environment. Actual stress placed on the system during active operation may be required to produce the fault, since it may be difficult to anticipate a test case or may be too expensive to simulate off-line. Also, these procedures impose a latency between the fault-occurrence and diagnosis phases, thus restricting the applicability of the diagnosis operations to actual system operations. Off-line techniques also leave the system susceptible to subsequently occurring faults in the time period the diagnosis for the prior fault occurrence is being addressed. Furthermore, off-line techniques are often invasive rather than self-contained, requiring additional resources to inject test sequences and monitor responses. It is worthwhile to mention that a number of response critical applications, the time to conduct an off-line fault diagnosis and recovery for transient/intermittent faults may simply be unacceptable.

Although on-line diagnosis has some clear benefits, it also has its challenges. The major challenge is to implement a procedure with high resolution that does not severely degrade the performance of the system. Approaches, such as the PMC [21] model (and variations - centralized and distributed), which need to explicitly schedule tests for obtaining syndrome information, are too costly in this regard and are, thus, invariably off-line procedures. Other proposed methods, such as comparison testing [28] also have their on-line shortcomings with the reliance primarily on comparison of test results for fault detection and not complete FDIR.

1.1 Our Diagnostic Approach

Our overall approach and contributions cover two specific areas:

- We develop on-line algorithms on a progressive basis, i.e., each developed algorithm provides coverage to a larger set of faults. The final algorithm achieves the capability of covering a complete range of faults from benign fault to faults of an unrestricted nature - all within the same algorithm. Furthermore, the algorithms are designed for flexibility to be directly customizable for a chosen fault model in an application. Unlike the assumption in classical approaches that diagnosis is conducted in a fault-free off-line environment, our algorithms provide on-line diagnosis and are themselves tolerant to faults in the diagnostic processes.

- We develop a formalization of the specification and verification of each of the developed algorithms. These algorithms are based on the interactive consensus approaches which have been demonstrated through earlier work on formal techniques to be particularly complex to analyze and susceptible to error in analysis. Thus, to provide a formal level of rigor in the correctness of the proposed algorithms and also to offer insights into the algorithm operations which arise from a rigorous formal analysis, we include formal verification of the algorithms as an integral part of our contributions.

From a fault model perspective, the classic stuck-at-X (fail-silent, fault-stop etc) model which assumes all faults to be benign and detectable usually results in overly optimistic evaluations. In reality, faults which exhibit less predictable manifestations are frequently present. In contrast, the Byzantine fault model assumes worst-case, unpredictable behavior for each fault occurrence. This

is a overly conservative model and results in a pessimistic assessment of dependability. The need, and our intent in this paper, is to find a more realistic and flexible fault model and to demonstrate its applicability in supporting generalized fault-diagnosis.

The majority of diagnostic approaches assume permanent faults to prevent syndrome ambiguities due to arbitrary test results from a node with a transient or intermittent fault. Thus, they are unable to handle Byzantine faulty nodes, which can send conflicting information to testing nodes. In fact, it is shown in [31] that identification of a Byzantine faulty node is impossible using a single round of test syndrome collection among the nodes. The authors in [31] do provide an off-line algorithm to tolerate Byzantine faults. In [35, 36], we have presented an on-line diagnosis and fault identification model that admits Byzantine faults. In [32, 36], we showed that using a more flexible hybrid fault-effect model(HFM) allows dependability to be more precisely and realistically evaluated for a given *fault set*[2] instead of a single fault type.

We deviate from the conventional fixed severity (both time-domain and data-domain) fault models to develop a composite and cohesive fault classification basis. Our model handles a continuum of fault and error types, classified according to their impact on system operations. Although this approach is shown to provide better assessments of operational indices, its practical usage requires an on-line FDIR process. In this paper we present diagnosis protocols, developing on our previous work[35, 36]. In [35], we demonstrated that different error types could be detected, focusing on local detection primitives. We presented the basis for an on-line algorithm which handles mixed fault types. One important feature shown was that this approach could be implemented with low overhead. It was stated that in MAFT[37] the overhead was $\leq 6\%$ even though the algorithm executes continually. This method allows diagnosis to be directed to identifying the cause of detected errors and their impact on system resources. Also, understanding the cause and effect relationship, allows the reconfiguration process to be properly guided.

Our initial classification of detectable faults as either benign or value faulty adopts a low-level perspective on information flow in the system. Extending beyond that approach, we provide a higher level classification from the standpoint of how detected faults affect the global state of the system based on the fault-effects model. This reconciling of local views to a higher-level global view is accomplished by first executing a distributed agreement algorithm to ensure consistent error syndrome information and penalty counts. This agreed upon information is used in the diagnosis phase by mapping low-level syndrome bits to high-level fault manifestation groups, such as benign, catastrophic, value, crash and Byzantine, which match the dependability model. Thus, we show that a more precise model for dependability can be constructed with significant benefits, supported by the requisite on-line algorithms. Our approach does not treat diagnosis as a stand-alone function but keeps it as an integral and continually on-going phase of system operations. Thus, the algorithm is not scheduled, but uses the system communication traffic as test stimuli. If necessary, specific tests can be run to handle episodic faults.

We detail our approach to the on-line diagnosis problem in an incremental manner where we successively develop algorithms which expand and generalize the applicable fault models (Section 2.2). In Section 2.2.1 we extend the fault-models further to consider diagnosis under the Hybrid Fault-Effects Model (HFM). Section 3, introduces our diagnostic approaches starting with a basic system model termed *PP* which assumes the absence of links faults and considers only benign fault occurrences. Sections 3 and 4 presents the diagnosis algorithm, and develop the perspective on

[2] i.e., a combination of various fault types

analysis of a diagnosis algorithm and also the process for formal specification and verification approaches for these diagnostic algorithms. After setting the analytical and formalization framework, we extend the diagnostic approaches for varied fault models. The *PP* model is enhanced in Section 5 to include considerations of arbitrary operational behavior of the nodes and also the communication links in the PLP model. Sections 5 and 6 discuss the new *PLP* model, the associated diagnosis algorithm and its analysis.

After developing these preliminary models to aid in the introduction of key concepts necessary for on-line diagnosis, we further generalize the approach to encompass the full scope of fault set coverage under the hybrid fault-effects model (HFM). As our algorithms are based on agreement protocols, which do not hold in the classical sense under the HFM, we need to develop modified agreement protocols and also conduct their verification. These aspects are discussed in Section 7. We present the progressively refined diagnosis algorithms (DD and HD) integrating consensus and diagnosis solutions under the HFM in Sections 8 and 9. In Section 10 we introduce the issue of the temporal behavior of faults through a notion of fault decay time and conclude the paper with a discussion of the developed techniques.

1.2 The Role of Formal Techniques

An important contribution of this paper is to associate formal arguments for the on-line diagnosis algorithms. For each developed algorithm, we present formal specification and associated formal verification. We emphasize that the coverage of Byzantine faults necessitates algorithms and proofs of considerable complexity, both obvious and subtle. Formal methods present a level of rigor and establish confidence in such proofs which is often lacking in hand proofs [27]. The intent being to present a composite and comprehensive solution to on-line diagnosis of arbitrary faults.

It is interesting to note that as we undertake the formal specification and verification of the algorithms, these processes sometimes generate insights which simplify the actual algorithm (e.g., F-PP in Section 4), facilitate better understanding of the rationale and also simplify the analysis.

2 System and Fault Model

2.1 System Model

We consider a distributed system framework, comprised of a completely connected network of *nodes* that communicate using a frame based and deterministic message passing protocol. Essentially, a frame based communication model implies that periodically, i.e., at the frame[3] boundaries, messages are sent and received by system nodes. The message delivery times are deterministic, providing bounded delays, and individual nodes make decisions and compute values based on information received in messages from other nodes. We consider the system communication model as:

A1: A direct path exists from each node to all other nodes (i.e., a completely connected network). A node issues a *single* message which is broadcast to all nodes connected to it.

[3]The terms "frames", "rounds" and "periods" are used synonymously.

A2: A non-faulty node can identify the sender of an incoming message, and can detect the absence or time-deviance[4] for an expected message. A frame based (synchronous), non-authenticated[5] message passing framework is assumed.

A3: All processors execute the same workload and determine the output value V_i through the use of a voting function.

A4: Node and link faults are considered indistinguishable[6]. Links are considered as simple memoryless interfaces between nodes.

A processor has a single transmitter which broadcasts the message to dedicated receivers. The protocol is non-authenticated message passing; however, as distinct communication channels are defined between processor-pairs, this results in a pseudo-authenticated paradigm where a faulty processor is prevented from corrupting a valid message of another processor without detection. In addition, error control checks (ECC) are performed on every message; thus, a malicious node would need to successfully: (a) forge the appropriate processor identity associated with this link, (b) compensate bit errors to bypass the ECC check, (c) do this at a time when the correct transmitter is silent, and (d) within the stipulated frame boundaries. Assumption A2 prevents a faulty processor from introducing spurious messages without detection, and from disrupting the decision process by failing to send a message. The use of a frame-based synchronous model also aids in identifying deviant messages received earlier or after the expected tolerance window at the frame boundary. Assumption A2 is implemented via the continuous on-line checking performed by each node. Finally, assumption A3 is implemented with a median select (or strict majority) voting algorithm which guarantees that, given a majority of non-faulty inputs, the error will be masked [37].

2.2 Fault Models

Our goal is to identify faulty nodes and to prevent system failure in the presence of a specified number of faults. The system fails when consistent decisions or computations across the system are no longer possible. Unlike previous work, we distinguish faults by their behavior, and place no limitations on the duration of a fault. Under this framework, the sole indicator of a faulty node is an error in its transmitted message, as viewed by all receivers of the message. Each receiver acquires a *local view* of the sending node's health by applying fault detection mechanisms locally, such as range and framing checks, to the incoming message. The exchange of local views among receiving nodes is then used to acquire a global perspective of the effects of a faulty unit.

We will utilize fault models which are flexible to cover a range of fault types. This is done to incorporate a realistic system environment where faults do not follow idealized fault conditions of only a single specific fault occurring over system operations. Initially, we start with considering a fault model termed PP which considers benign processor faults. Extending beyond the PP we consider the fault models to handle arbitrary fault conditions as well as communication link failures

[4] messages from a processor are sent (or received) during a time window specified around the frame boundaries.

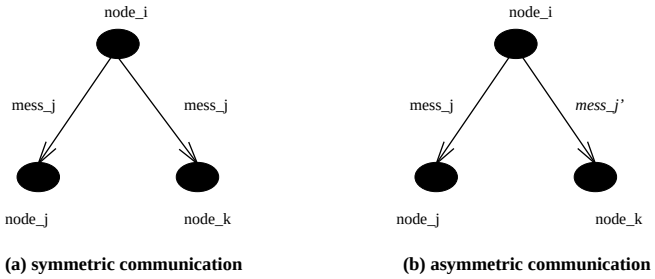
[5] authenticated techniques involve association of a signature with a message which explicitly characterizes its identity to prevent another node forging this message. The authenticated approach involves high bit-space and time overhead and most implemented systems choose the simpler non-authenticated approach.

[6] This assumption is later alleviated in Algs. PLP and HD.

within a single diagnostic approach. This model is termed as the PLP model. After introducing the conceptual diagnostic framework under the PP and PLP models, we generalize the fault models to a very flexible and customizable fault model termed as the Hybrid Fault-Effects Model(HFM) which we have introduced and presented in earlier work [32, 36]. To facilitate discussions based on the HFM, we provide a summary introduction of this fault model.

2.2.1 The Hybrid Fault-Effect Model (HFM)

Our earlier work [32, 36] details the properties of the HFM and its properties in detail; we provide a brief introduction of the model here to facilitate discussions. In the HFM, all fault classification is based on **(1)** *Local-Classification* of fault-effects to the extent permitted by the fault-detection mechanisms built in at the node level, and **(2)** *Global-Classification* based on nodes exchanging their local-classification with other system nodes to develop a global opinion on the fault-effect. We now describe in greater detail the rationale behind the individual classifications.



	locally detectable	*globally detectable*
symmetric communication	**Benign**	**Value**
asymmetric communication	**Benign**	**Value** (arbitrary faulty)

Figure 1: Hybrid Fault Space (communication symmetry vs. detectability)

The HFM classification begins with the examination of the possible fault states created by the dispersal of information. In the abstract, we can assume that when a node transmits information, two general cases are possible as illustrated in Figure 1(top): **(a)** all receivers obtain the same information i.e., $mess_j$ *(symmetric dispersal)*, or **(b)** receivers obtain different information i.e., $mess_j$ and $mess'_j$ (*asymmetric dispersal*). This abstraction is sufficient since we are primarily concerned with maintaining consistency and these two possibilities reflect whether it is preserved or not.

The second aspect of the HFM is based on extent of the fault-effect as created by the faulty source and the way in which it has propagated its effects. If an individual node is sufficient to detect the error, we classify this case as a *benign fault* since the fault-effect is *locally detectable*.

On the other hand, there are situations that require multiple nodes to exchange their syndrome information with each other in order to provide accurate diagnosis, i.e., *globally detectable*. For these cases, the values in a message appear to be plausible locally at a node; however, they can only be verified with a multiple exchange of information. After the exchange is completed, the plausible value may be determined to be *value-faulty*.

As an example, consider the acceptable range of the value contained in a message to be [0,100], with the expected correct value of 50 in a message. A node accepts a message without registering any fault as long as the value is within [0,100]. If a faulty node sends values 30 and 70 as values in the messages (instead of the correct value of 50) to the other two nodes, both of these messages are within the acceptable range [0,100], thus these erroneous values are not locally detectable at the receiving nodes. Only when the two receiving nodes exchange (or distribute) their received message values of 30 and 70 respectively with each other, does the asymmetric nature of the faulty values comes across i.e., global detectability of value-faults.

Figure 1 summarizes the specific cases covered by the Hybrid Fault Model. The vertical axis reflects the possibilities for information dispersal and the horizontal axis represents the types of detectability. The fault-effects are represented by the table entries and consists of two major types covered in each vertical column: benign and value faults. The *benign* faults can be dispersed in a symmetric or asymmetric manner; however, these faults are detectable by all non-faulty nodes with fault detection mechanisms implemented locally. Any detectable fault in a message will result in the sender being classified as *locally benign faulty, (b),* by the receiver. The rightmost vertical column of the table covers *value* faults which are the locally undetectable class of faults. Messages which pass all data validity and range deviance checks, but provide a valid but *incorrect* value constitute value faults. Since the effect of these faults are dependent on the type of information dispersal, we have the sub-cases of *symmetric-value faults, (s),* and *asymmetric-value faults, (a),*. The asymmetric-value fault is equivalent to the case of a Byzantine fault and is also referred to as the *arbitrary fault* case.

Diagnosis of value faults is more troublesome to handle[7] and the categorization of faults as symmetric or asymmetric requires a global determination, or several rounds of message exchange. Suppose at least one non-faulty node cannot detect a node fault, and the same fault is detected by every other good node. The sending node is identified as *locally benign* by the nodes that can detect it but the local view is not sufficiently accurate to provide correct system level diagnosis. Thus, exchange and accumulation of global information across several time periods is often necessary to diagnose such faults.

Byzantine agreement [11] algorithms are often implemented to ensure that distributed nodes or processes arrive at the same decisions and computational results in the presence of arbitrary faults. However, with simple modification of existing fault detection and masking techniques, correct computations can also be guaranteed under the HFM. Unlike the $N > 3t$ redundancy requirement of the classical Byzantine models to sustain t faults, under the HFM N nodes will tolerate $> 3a + 2s + b$ composite *(a, b and s)* faults. We extend the classical Byzantine fault model to address mixed fault types, while ensuring correct system operation in the presence of faults. Instead of assuming that all faults are *arbitrary*, we adopt the hybrid fault-effects (HFM) taxonomy and handle faults according to the errors they produce.

[7] In some prior work we have used the term *malicious* to highlight this difficulty.

The following Table 1 summarizes our development of the diagnosis models as per the different fault classes considered in this paper.

Table 1: Association of Developed Algorithms and Fault Models

Fault Class	**Associated Algorithm**	**Section**
Benign processor faults (No link faults)	PP	Sections 3 & 4
Arbitrary processor faults and Link faults	PLP	Sections 5 & 6
–Foundation of Consensus under HFM–	OMH	Section 7
HFM Model (No Temporal Aspects)	DD	Section 8
HFM Model Temporal Aspects & Complete FDIR	HD	Sections 9 & 10

2.3 Terminology

We introduce the basic terminology which will be used (and progressively expanded) throughout the the paper. We consider N as the number of processors in the system and $mess_j$ representing a message sent by processor j. As the communication model is frame based i.e., messages are sent/received by nodes at the frame boundaries. Thus, the frame (or round) number is also a useful component in identifying a message. Let $\mathcal{M}_i^n(j) \forall i, j$ define the set of all $mess_j$, sent by processor j over frame n, as received by processor i. As the node communication is based on exchange of messages, and as we categorize faults by their perceived fault-effect as seen by the recipient of the message, we introduce fault categories for the messages based on the receiver's observations on these messages. Initially we consider two such fault categories: (a) The set of *missing messages*, $MM_i^n(j)$, are those messages which i believes j failed to issue during frame n, and (b) The set of *improper logical messages*, $ILM_i^n(j)$, are those messages which are correctly delivered but disagree with V_i, the result of i's own voting process on inputs received. The *syndrome* $\mathcal{S}_i^n(j), \forall i, j$ represents the union of $ILM_i^n(j)$ and $MM_i^n(j)$. $\mathcal{S}_i^n(j)$ is represented in vector form for each value of i, with vector entries corresponding to all j values from which i receives messages. The vector entry corresponding to any node j is a binary input: 0 corresponds to a fault-free input received from j as perceived by i, and 1 represents a fault being perceived by i.

Each node maintains its perception of the system state using a system level error report in the form of a vector, $F_i^n(j)$[8], consisting of an ordered quadruple $< i, j, n, \mathcal{S}_i^n(j) >$. The function $F_{tot}^n(j) = |\bigcup_{i \in N, i \neq j} F_i^n(j)|$ is used to count the number of accusations on a processor j by all other processors during frame n. Thus, $F_{tot}^n(j)$ is an integer where $0 \leq F_{tot}^n(j) \leq (N-1)$.

For all our developed diagnosis algorithms, the underpinnings will involve variations of a two-phase diagnostic approach.

[8] as received by i from node j over round n

Phase 1: local diagnosis syndrome formulation based on a node's local perception of other nodes fault status. This is established through a node's personal analysis of incoming message traffic from other nodes.

Phase 2: global diagnosis syndrome formulation through exchange of local diagnosis syndrome information to all other nodes.

2.4 Comments on Approach

- For each algorithm we develop, we will highlight the fault model being considered and aspects of the two-phase approach as they apply to the fault model. We will then present the analysis of algorithmic properties followed by the development of the formal model of the algorithm and associated formal analysis and verification. We will utilize this format for all algorithms.
- For the initial algorithm PP, we will present an extensive formal development to illustrate the concepts involved. To a large extent, the formal analysis and verification of the subsequent algorithms PLP, DD and HD will utilize the formal infrastructure developed for algorithm PP. This highlights the aspects of re-usability of formal concepts as progressively more complex algorithms are developed – for diagnosis and for other algorithms in general.

3 Model PP: Diagnosing Benign Faults

We now present the first diagnosis algorithm termed PP. The initial model will be referred to as the *processor - processor (PP)* model since it will be assumed that all the communication links are non-faulty and that processors are the only potentially faulty units. As stated in Section 2.3, the two-phase diagnostic approach will characterize the operations of the algorithm.

Algorithm PP

D1.0 For all $i, j \in N$, each processor i monitors each $mess_j \in \mathcal{M}_i^n(j)$.

D1.1 If the value v_j contained in $mess_j$ does not agree with V_i[9], then $mess_j \in \mathit{ILM}_i^n(j)$,

D1.2 If $mess_j$ is missing, then $mess_j \in MM_i^n(j)$,

D1.3 Update the syndrome information: $\mathcal{S}_i^n(j) = \mathit{ILM}_i^n(j) \cup MM_i^n(j)$.

D2.0 At the completion of frame n, for every j, each i will determine if an error report should be issued: if $\mathcal{S}_i^n(j) \neq \emptyset$ then send report $F_i^n(j)$ as sourced by i to other processors, else do not send $F_i^n(j)$.

D3.0 For each j, as frame $n+1$ completes, compute $F_{tot}^n(j)$.

D3.1 If $F_{tot}^n(j) \geq \lceil N/2 \rceil$ then declare j as faulty.

D3.1.1 If processor k failed to report $F_k^n(j) \neq \emptyset$ then $mess_k \in MM_i^{n+1}(k)$

D3.2 If $F_{tot}^n(j) < \lceil N/2 \rceil$ then

D3.2.1 If k reported $F_k^n(j) \neq \emptyset$ then $mess_k \in \mathit{ILM}_i^{n+1}(k)$

D4.0 Increment frame counter n and proceed to step D1.

---

[9] V_i is the voted values - see assumption A3 in Section 2

The error detection process is summarized by step D1.0. During frame n, each processor monitors the messages received and performs error checking. The logical content errors identified in step D1.1 are detected by voting on the inputs and then checking the inputs against the voted value (i.e. deviance checking). Omissions of expected messages are also detected and recorded in D1.2. In step D1.3, these errors are written into a local error log to be processed at the completion of frame n. In step D2.0, if any errors have been logged, a system level report is issued accusing the suspected processor. These reports are counted in step D3.0 and the accused processor is declared faulty provided at least half of the system agrees on the accusation. The diagnostic processors are thus also checked as part of the algorithm. In D3.1.1, if j is determined to be faulty but a monitoring processor k failed to report an error on j, processor k will be accused as faulty in the succeeding round of diagnosis. In D3.2.1, if only a minority of processors accused j, they will be accused as faulty in the next round.

We stress that the number of processors, both operational and faulty, must be within the specified fault-tolerance processor cardinality as defined for the benign fault case. In general, this cardinality will be distinctively specified for the fault model being considered e.g., $N > 3t$ for t Byzantine faults.

3.1 Basic Analysis of PP

Prior to analyzing PP (or any diagnosis algorithm), it is essential to establish two specific properties of *correctness* and *completeness* that a diagnosis algorithm is required to satisfy [31]:

P1. *Correctness:* every processor diagnosed to be faulty by a non-faulty processor is indeed faulty.

P2. *Completeness:* every faulty processor is identified.

We will now show that algorithm PP satisfies both properties under the described PP model.

Theorem 1 *(Correctness) If "i" is non-faulty, then $F_i^n(j) \neq \emptyset$ implies that j is faulty ensuring the diagnosis of Algorithm PP to be correct.*

Proof: If the receiving node i is non-faulty and accuses j such that $F_i^n(j) \neq \emptyset$, then there exists a $mess_j \in \mathcal{M}_i^n(j)$ which was found to be in error. Two possibilities exist: (i) either another node k forged j's identity or (ii) j sent an erroneous message. Case (i) is precluded by A2 and especially since each processor-pair communicates over separate links. Thus, if $F_i^n(j) \neq \emptyset$, it will be because j is faulty. □

Theorem 2 *(Completeness) If "j" is faulty, $F_i^n(j) \neq \emptyset$, for every non-faulty $i \in N$, ensuring the completeness of diagnosis by Algorithm PP.*

Proof: There are two cases where j is faulty: (i) j sends erroneous $mess_j \in \mathcal{M}_i^n(j)$, or (ii) j fails to issue all $mess_j$. If $mess_j$ is delivered correctly, then the value v_j is compared with the receiving processor's voted result V_i. If $v_j = V_i$, then j has computed the correct value and is not faulty. If $v_j \neq V_i$, then i will be able to detect the erroneous message and $mess_j \in F_i^n(j)$. Case (ii) is restricted by assumptions A1 & A2, since an absence of a message will be detectable. Thus, for both cases, if j is faulty then $F_i^n(j) \neq \emptyset$. □

After defining the additional notation to be used, it will be demonstrated that $\lceil N/2 \rceil$ is a suitable threshold for determining if j is faulty.

Definition P_{min} is the minimum number of processors needed to guarantee correct diagnosis of a faulty j.

Definition P_{max} is the maximum number of processors that perform correct diagnosis of a faulty j. This term is better understood as $P_{max} = N - t$ for t faults.

Definition The *error support threshold*, E, is the threshold implemented in the diagnosis algorithm to determine if processor j is faulty.

Lemma 1 *If $N \geq 3$ and $F_{tot}^n(j) \geq E = \lceil N/2 \rceil$ then j is faulty.*

Proof: Suppose that t of the N processors in the system are faulty. Then, at most $(N - t)$ non-faulty processors will correctly accuse a faulty processor, giving $P_{max} = N - t$. To determine the number of processors needed to outvote the faulty units in step D3, we write $t = t_j + t_{mon}$, where t_j is a fault in the monitored processor, t_{mon} are faults in the monitoring processors i.e., good processors. An interesting feature of our algorithms becomes apparent here. Unlike classical diagnosis algorithms, we do not make any simplifying assumptions of conducting diagnosis in a fault-free off-line environment. As our algorithms are on-line, these are themselves exposed to system faults too. Thus, we explicitly consider our diagnosis algorithms to be tolerant to a specified number of fault, t_{mon} while diagnosis is ongoing.

Since j is always considered to be suspect and is not allowed to participate in the diagnosis on itself, it is not included as one of the possible faulty monitoring units. There needs to be at least two non-faulty processors reporting $F_i^n(j) \neq \emptyset$ to prevent false diagnosis (e.g. in a three processor system, at least two processors need to agree on an accusation). Furthermore, a minority of processors (other than j) can be faulty so that $P_{min} = max(2, t_{mon} + 1)$. The allowable range for a correct diagnosis is therefore $P_{min} \leq E \leq P_{max}$. Since the selected threshold $E = \lceil N/2 \rceil$ is within this range, it suffices as a suitable threshold for diagnosing j as faulty. □

Definition A *valid alarm* is issued by processor i if $F_i^n(j) \neq \emptyset$ and $F_{tot}^n(j) \geq E$.

Definition A *invalid alarm* is issued by processor i if $F_i^n(j) \neq \emptyset$ and $F_{tot}^n(j) < E$.

Even though P_{min} processors are sufficient to detect an error by the sender j, it should be expected that if error conditions are symmetric, P_{max} processors should agree with the diagnosis. This result provides the basis for verifying correct response of the diagnostic units. The selection of $m = \lceil N/2 \rceil$ provides a balance in tolerating faults in both the monitored and monitoring processors.

As an example we will assume a system containing six processors which are denoted $p_1,\ p_2,\ ...,\ p_6$ and their associated links are $\ell_1, \ell_2,\ ..., \ell_6$ respectively. Let p_1 be faulty during frame n. If p_1 is faulty, then by Theorem 2 we know that $p_2,\ p_3,\ p_4,\ p_5,\ p_6$ should all respond with valid alarms $F_i^n(1) \neq \emptyset$. If p_6 were a faulty monitor and did not send $F_6^n(1) \neq \emptyset$, then it would be accused as faulty in step D3.2.1 and the checking process would propagate into frame $n+1$ with $F_i^{n+1}(6) \neq \emptyset$. In the next frame, $p_2,\ p_3,\ p_4,\ p_5$ would agree $F_i^{n+1}(6) \neq \emptyset$ and conclude p_6 is faulty. Thus, the property of completeness is used to guarantee that all the monitors functioned properly.

The property of correctness can also be used to check the checkers. Assume p_6 is a faulty monitor and accuses p_1 such that $F_6^n(1) \neq \emptyset$. If p_1 is non-faulty, then Theorem 1 guarantees that all non-faulty monitors will not detect a fault such that $\forall\ i \in N, i \neq 6, F_i^n(1) = \emptyset$. Monitor p_6 therefore issued an invalid alarm and will be accused as faulty in step D3.2.1. In frame $n+1$, non-faulty processors will respond with $F_i^{n+1}(6) \neq \emptyset$. In the next frame, $p_1, p_2,\ p_3,\ p_4,\ p_5$ would agree $F_i^{n+1}(6) \neq \emptyset$ and conclude p_6 is faulty.

4 Developing a Formal Analysis for Algorithm PP

The presentation of Algorithm PP is consistent with traditional forms of hand proofs extensively found in literature [4, 9, 10, 11, 12, 15, 20, 21, 22, 29, 31, 34, 35]. It is not, however, directly usable for input to a theorem prover. The initial form, PP, contains supplementary information which although of interest to the system designer, is unnecessary and distracting to the formal proof process.

In order to facilitate formal analysis, we first rewrite algorithm PP in a simplified form, termed F-PP, reducing the notational complexity and emphasizing the operations performed. We point out that the informal and formal representations of the algorithms are equivalent. The basic intent of rewriting of the algorithm is to aid in focusing on the core properties we want to incorporate and verify in the formal specification of the diagnosis algorithm, rather than detail the data structures involved.

Algorithm F-PP
PP(0)

1. All accusations of faults are cleared.

PP(n), n > 0

1. Each processor i executes one frame of the workload, arriving at some value or checksum $Val^n(i)$.
2. Each processor sends $Val^n(i)$ to all other processors.
3. Each processor i compares incoming messages to its own value:

(a) If the value from j does not match, is missing, or is otherwise detectably benign, or there is an accusation from the last frame of i against j, record BAD.

(b) Otherwise, i records that j is GOOD.

4. Each processor sends its report on each other processor to all processors.

5. Each processor i collects all votes regarding each other processor j:

(a) If the majority of votes are BAD, then processor i declares j faulty. Furthermore, i records an accusation against any processor k that voted j GOOD.

(b) If the majority of votes are GOOD, then i records an accusation against any processor k that voted j BAD.

In this rewriting of the algorithm, the initial frame, referred to as PP(0), simply initializes the data structures appropriately. Next, a workload frame is executed (Step 1), arriving at some value. Processors then exchange values (Step 2). All good processors should then have exchanged identical values. Faulty processors may have exchanged corrupted values that are locally detectable, but we do not consider the possibility here that faulty processors deliver different values to different receivers. All processors then compare the exchanged values with their own. Any discrepancy is recorded as an accusation against the sending processor.

The key assumption here is that if any processor is symmetrically bad in a locally detectable manner, then all receivers must have gotten bad messages from that processor. Therefore if the majority of processors have seen erroneous values from j, then j is in fact bad.

4.1 Discussion

The formal specification, F-PP, clarifies the key aspects of the original description of PP algorithm. For example, in place of "if the majority of the votes are BAD," the original PP uses $F^n_{tot}(j) \geq \lceil N/2 \rceil$, where $F^n_{tot}(j)$ is the number of accusations from processors other than j. Since the algorithm is not concerned with the internal behavior of faulty nodes, and good nodes never accuse themselves, in F-PP we simply count the total number of accusations. This kind of simplification is motivated by our desire to remove complexity from the formal specification and verification process wherever it would entail a high cost. Accordingly, we endeavor to simplify algorithms before formal tools and techniques are applied.

Another aspect is that the original PP separates bad messages into categories ILM and MM, and then combines these sets. F-PP simply records GOOD or BAD. The original algorithm PP states that processors send accusations *signed* by the sender. F-PP assumes the existence of private channels between all processors, so that a message is implicitly signed by virtue of arriving on that private channel. In both cases it is assumed that messages sent directly from one processor to another cannot be forged. However, since messages are not forwarded from one processor to another, neither PP nor F-PP make any assumption about the possibility of forging forwarded messages. It is worth mentioning that the original specification of PP and the specification of F-PP used to build the formal model amount to the same algorithm. Since the benefit of formal verification depends on the accuracy of formal specification, it is very important to maintain fidelity to the PP operations while formulating F-PP. This process helps in obtaining additional insight into the overall algorithmic operations and facilitates better understanding of the algorithm.

Algorithm PP operates effectively under the assumption that all faults are permanent within a frame, but does not rely on the consistency of faults between frames. This kind of inconsistent behavior within a frame is handled only by Byzantine resilient algorithms such as DD (or HD), analyzed later in Section 8. However, a processor that fails symmetrically, say, for just one frame and is otherwise non-faulty, would be correctly diagnosed as faulty in the frame it became bad.

For formal analysis, we split the presentation into (a) a general argument which builds up the formal context, followed by (b) the detailed formal analysis. This paper will include the development of the formal arguments and specification of each algorithm in the main part of the paper and defer the details of the formal specifications to the Appendix.

4.2 Formal Analysis: Developing the Formal Argument

Formally, the requirements follow directly. First, we consider correctness, which states that if a good processor accuses some other processor, the accused processor is indeed faulty.

Lemma 2 *If "i" is good, then $F_i^n(j) \neq \emptyset$ implies that "j" is faulty.*

Proof: We prove this by complete induction on the number of rounds of algorithm PP we use. If $F_i^n(j) \neq \emptyset$, then some message sent from j to i was bad. If the message was simply missing or was an improper logical message, then j is in fact bad. The two remaining reasons that i would record an accusation of j is if j failed to accuse some bad processor, or if j accused some good processor in an earlier frame. By induction we can assume that the previous majority votes on the fault status of other processors are correct. Since all faults are assumed to be symmetrically dispersed, if j failed to accuse a bad processor k, then j is bad, as all processors including j must have received the same bad message from k. If j accused a good processor k, then j must be bad since no majority of good processors received a bad message from k. □

Theorem 3 *(Correctness) If a good "i" declares "j" faulty, "j" is indeed faulty.*

Proof: We prove this by complete induction on the number of rounds used by algorithm PP. The only place in the algorithm that a good processor could declare another processor faulty is when $F_{tot}^n(j) \geq \lceil N/2 \rceil$. This can arise only if over half of all processors send an accusation to i. Since we assume that over half of all processors are non-faulty, some good processor k must have contributed by sending an accusation of j to i. By Lemma 2, we thus know that j is indeed faulty. □

This formal argument does add some detail to the original definition of algorithm PP. Even so, explicit formalization of the algorithm does not follow directly. For example, there is no basis on which to make the assertion that "by induction we can assume that the previous majority..." as the inductive hypothesis is not about majorities. A flavor of completeness is inductively being assumed here. It happens that full completeness is not needed in the proof of soundness, but there is more to the formal proof than is apparent from these general proofs.

Theorem 4 *(Completeness) If "j" is faulty, then all good processors diagnose "j" as faulty.*

Proof: We prove this by complete induction on the number of rounds used in PP. For round n, we assume that for smaller number of rounds this property holds. If j is faulty in earlier rounds then by induction we can assume that i declared j faulty earlier, and thus since declarations are monotone, i declares j faulty. If j is faulty in this round, it must be either benign faulty, in which case all messages are benign, or symmetric-value faulty. If the symmetric-value faulty processor produces a good value then it is not a fault. Thus even symmetric-value faulty processors must produce bad values. In either case all good processors will accuse j of being faulty, sending $F_i^n(j)$ to all other processors. Since we assume there are a majority of good processors, there will be a majority of votes to condemn j, and all good processors will declare j faulty. □

Even with these additional refinements to the description of the algorithm, this level of detail still falls short of that needed to complete full formal verification. We have formally specified the PP algorithm and formally verified that it satisfies both of its main requirements using the PVS verification system [17, 18, 19]. We describe this formal specification and verification in the next sections.

4.3 Formal Analysis: Developing Formal Specification of PP

The formal specification of PP is embodied in a single PVS theory called *pp*. This theory takes several parameters, the key ones of which are m, the maximum[10] number of rounds, n, the number of processors, and T, the type values that are passed between processors. Although a few assumptions are made about T, it is mostly left undefined, and thus we effectively make no assumptions about the kinds of values passed around the system. The other parameters to the theory are some special values in the set T. Thus, *error* represents values that are benign upon local receipt, such as missing values, values failing parity check, values failing digital signature checks, and so on. *BAD* and *GOOD* are the values of accusations sent by processors over the network. Finally, the function *Val* is assumed to return the correct value for each frame of computation, and that the correct value is never any of the special values *error*, *BAD*, or *GOOD*.

After some definitions of types and variables are given, some other predefined theories are explicitly imported. The theory *filters* defines a function *filter* that returns the set of members of a given set that satisfy a given predicate. Since predicates and sets are equivalent in higher-order logic, this operation is the same as set intersection.[11] The theory *card_set* provides some standard lemmas concerning cardinality, and filters (for example, the cardinality of a set is nonzero if and only if the set is nonempty); it takes the same arguments as *finite_cardinality*. The imported theory *hybridmjrty* is not essential to the main development here and is described in [13].

The type *statuses* is defined to be an enumeration of three constants, corresponding to three of the categories of behavior: *symmetric-value faulty*, *benign*, and *good*.[12] In later specifications of other algorithms, the other two formal *statuses* will be represented by extended statuses to include *arbitrary* and/or *link*.

[10] This provides a bound on the time to actual diagnosis.

[11] The theory *filters* also provides a similar function on lists, which is rather more complex.

[12] Enumeration constants also generate recognizer functions in PVS. Thus, if s is a variable of type *statuses*, then $s = arbitrary?$ and $arbitrary(s)$ are equivalent formulas. The latter form is used in this specification.

The function *status* returns the status of a given processor (or fault containment unit *fcu*); this implicitly enforces the notion that a processor may not change status during execution of the agreement protocol. A processor that, in reality, is symmetric-value faulty one moment, benign the next, and good the next must be modeled as one that is symmetric-value faulty throughout the computation.

Some shorthands are then defined for describing statuses: s, c, and g are predicates recognizing the symmetric-value faulty, benign, and good processors, respectively (c stands for crash-faulty, a predominant cause of benign faults: m is not used to avoid confusion with the traditional use of m as the number of rounds.) Similarly, given a set *caucus*, $as(caucus)$ is the set of arbitrary-faulty processors in *caucus*. The functions *ss*, *cs* and *gs* similarly select the symmetric-value faulty, benign, and good processors, respectively. A simple lemma, *fincard_all*, states that the cardinality of a set of processors is equal to the sum of the cardinalities of the subsets of its processors of each status. This lemma follows from a property implicit in the definition of statuses as an enumeration type: the members of the enumeration are inclusive and disjoint.

The function *send* captures the properties of sending values from one processor to another. This function takes a value to be sent, a sender, and a receiver as arguments; it returns the value that *would be* received if the receiver were a good processor. The result actually received is irrelevant if the receiver is not a good processor (because the values passed on by faulty receivers are not assumed to be related to those received). The behavior of *send* is axiomatized according to the status of the sender. The first axiom simply says that a good processor sends correct values to all (good) receivers:

$$g(p) \supset send(t, p, q) = t.$$

Here, and in further formal definitions, free variables are universally bound at the outermost level, and the types of all variables are omitted for brevity. See the complete specification below for subsidiary and variable declarations. The second axiom says that a benign faulty processor always delivers values that are recognized as erroneous by good receivers:

$$c(p) \supset send(t, p, q) = error.$$

The third axiom says that a symmetric-value faulty processor sends the same value to all good receivers, although that value is otherwise unconstrained (i.e., it may be any possible value, including those that are recognized as erroneous)

$$s(p) \supset send(t, p, q) = send(t, p, z).$$

Nothing is specified for the behavior of asymmetric-value faulty senders. A lemma (called *send5*) is stated and proved that all receivers obtain the same value no matter what the status of the sender (remember that in this theory the possibility of link and arbitrary faults is discounted)

$$send(t, p, q) = send(t, p, z).$$

A deficiency of this specification is that, because *send* is a function, even arbitrarily faulty processors are consistent from one round to the next: the value $send(t, p, q)$ is some fixed value, suggesting that a faulty processor p, given the same value t, will always send the same (possibly bad) value to the processor q, even in different rounds of the protocol. This fact is not exploited in

the proof, but it is not self-evident that this is so. In the verification of the OM algorithm [24], the addition of the round number as an additional argument was done to *send* in order to lessen this concern. However, the only way to allay such doubts absolutely is to specify *send* as a relation. In earlier work the OM algorithm was axiomatized using a relational *send*, and the corresponding correctness conditions were proven. Unfortunately, the relational *send* complicates and obscures the specification (since it forces other functions to become relations also), so we have chosen to retain a functional *send* for the current specification.

The function *HybridMajority* is intended to be similar to the standard *Majority* function, except that all *error* values are excluded. Here we give an axiomatization of this function rather than a definition. This allows us a certain freedom in the implementation of this function. Although the three properties are all that is required of an implementation of *HybridMajority*, in earlier work a concrete implementation of *HybridMajority* was provided based on the Boyer-Moore MJRTY algorithm [3], and proved that the axioms below are satisfied by this implementation [13]. Thus the following may be considered axioms, or may be considered lemmas proven by appeal to imported *hybridmjrty* theory:

```
  HybridMajority(caucus, v): T = proj_1(Hybrid_mjrty(caucus, v, n))

  HybridMajority1: LEMMA
    fincard(gs(caucus)) > fincard(ss(caucus))
        AND (FORALL p : g(p) AND member(p,caucus) IMPLIES v(p) = t)
AND t /= error
AND (FORALL p : c(p) AND member(p, caucus) IMPLIES v(p)=error)
      IMPLIES HybridMajority(caucus, v) = t

  HybridMajority2: LEMMA
    (FORALL p : member(p, caucus) IMPLIES v1(p) = v2(p))
IMPLIES HybridMajority(caucus, v1) = HybridMajority(caucus, v2)

  HybridMajority3: LEMMA
    HybridMajority(caucus, v) = t
        AND (FORALL p,q : g(p) AND g(q) AND member(p,caucus) and member(q,
caucus)
IMPLIES (v(p) = v(q) AND v(p) /= error))
AND fincard(gs(caucus)) > fincard(ss(caucus))
AND (FORALL p : c(p) AND member(p, caucus) IMPLIES v(p)=error)
     IMPLIES
 (FORALL p : g(p) AND member(p,caucus) IMPLIES v(p) = t)
```

The first line of this fragment of the specification can be read as follows. The function *HybridMajority* takes two arguments, a set of processors (i.e., an *fcuset*), which we call the *caucus*, and a vector mapping processors to values (i.e., an *fcuvector*).

The antecedent to the implication in the second clause of this specification is complicated, but can be read as follows. If the vector records the same non-error value for all good processors in

the caucus, and the vector records an error value for all benign-faulty (benign) processors in the caucus, and there are more good processors than symmetric-value faulty processors in the caucus, then *HybridMajority* returns the same value as that recorded in the vector for the good processors. Any implementation of *HybridMajority* that does in fact compute the true majority after casting out error values would satisfy this axiom.

The third clause, *HybridMajority2*, states that the value returned depends only on the values recorded in the vector for the processors in the caucus. Although *HybridMajority* is a function, it could potentially be implemented in such a way that when there is no majority (i.e., when the antecedent to the implication above is false), the output depends on values of the vector corresponding to processors not in the caucus, or other irrelevant information contained in the arguments. The second axiom prohibits this kind of behavior.

The fourth clause, *HybridMajority3* is given for convenience. It states that if there are more good than symmetric-faulty processors and all good processors agree on some non-error value, and the *HybridMajority* function returns a value, then that value is the value of each good processor. This is a direct consequence of *HybridMajority1*, but is added to ease use in formal verification.

We then begin definition of the actual algorithm.

```
Syndrome(R,j,i,OldAccuse): T =
  IF OldAccuse(i,j) OR (NOT Val(R) = send(Val(R),j,i))
  THEN BAD
  ELSE GOOD
  ENDIF
```

The *Syndrome* function above is meant to capture the property that in round R, i believes j is faulty. The parameter *OldAccuse* essentially records old accusations from earlier rounds (As we will see later, in algorithm PP, if a processor i does not accuse a bad processor k, then all other good processors j will accuse i of being bad the next round.) The only other reason to accuse a processor of faulty behavior is if that processor sent some value that does not correspond to the correct value. The next function is built using *Syndrome*

```
KDeclareJ(pset,R,OldAccuse,j,k): bool =
   HybridMajority(pset, LAMBDA i: send(Syndrome(R,j,i,OldAccuse),i,k))=BAD
```

This predicate is meant to capture the idea that processor k will gather all accusations against some processor j, and then take the *HybridMajority* of that set. If most processors accuse j, then this predicate is true. Next,

```
PP(pset, R, OldAccuse)(i,j): RECURSIVE bool =
 IF R = 0
 THEN FALSE
 ELSE KDeclareJ(pset,R,OldAccuse,j,i)
      OR
      PP(pset, R-1,
 (lambda i2,k: OldAccuse(i2,k) OR
```

```
    EXISTS j2: (KDeclareJ(pset,R,OldAccuse,j2,i2)
       /=
       (send(Syndrome(R,j2,k,OldAccuse),k,i2)=BAD))))(i,j)
  ENDIF
 MEASURE (LAMBDA pset, R, OldAccuse -> nat : R)
```

Here we have the top-level algorithm *F-PP*. As stated above, this most closely corresponds to the algorithm PP and captures essentially the same process as rewritten in the formal algorithm F-PP. The intended meaning of this formal description is that after R rounds, starting with *OldAccuse* accusations, processor i believes that processor j is faulty. *PP* is defined as a recursive function. If the number of rounds R is zero, then i will not accuse j. If *KDeclareJ(pset,R,OldAccuse,j,i)*, that is, if after gathering votes for round R, a (hybrid) majority of other processors send i an accusation of j, then i believes j is faulty. Otherwise, *PP* is called recursively, using one less round. The recursive call also updates *OldAccuse* to include the case that some processor misdiagnosed some other processor. That is, an accusation is added to the local *OldAccuse* for the next round if the voted diagnosis *KDeclareJ(pset,R,OldAccuse,j2,i2)* of some processor $j2$ does not agree with the individual accusation sent from k to $i2$.

The remainder of the specification consists of interesting properties of the PP algorithm. Many of the following theorems are first defined as predicates, then a lemma asserting that this predicate is universal is proved by induction on the number of rounds, and then a theorem giving the result in the form desired is stated. This style of breaking a specification into separate predicate, lemma, and theorem is quite useful in formal systems. Many other large specifications use this technique [30, 40, 13].

We now present the final theorem versions of the two properties.

```
Final_Correctness: THEOREM
(FORALL i,j:
    g(i)
AND fincard(gs(fullset[fcu])) > fincard(ss(fullset[fcu])) + 1
AND PP(fullset[fcu],R,Empty)(i,j)
IMPLIES
    c(j) OR s(j))
```

This theorem is supposed to roughly correspond to the first formal requirement: "**Correctness**: If PP says a processor is faulty, then it is indeed faulty." Formally, we must make explicit the assumption that there are enough good processors around, and what is meant by "PP says." The key property being addressed here is that all good processors accuse only faulty processors of being faulty. This formally becomes

```
 g(i) AND PP(fullset[fcu],R,Empty)(i,j) IMPLIES c(j) OR s(j)
```

which can be read as: "If i is good, and after R rounds of PP, i accuses j, then either j is benign or symmetric-value faulty." Formally, a condition needs to be added that there be enough good processors. The second theorem is

```
 Final_Completeness: THEOREM
(FORALL i,j:
g(i)
AND (c(j) OR (s(j) AND (FORALL t,p: send(t,j,p) /= t)))
AND fincard(gs(fullset[fcu])) > fincard(ss(fullset[fcu])) + 1
AND R>0
IMPLIES
           PP(fullset[fcu],R,Empty)(i,j))
```

This theorem tries to capture the following property: "**Completeness**: If a processor is faulty, then algorithm PP will determine this." Again, an extra assumption must be made that there are enough good processors. The statement is simple in the case of benign faults. In the case of symmetric-value faults, however, a further assumption must be made that the symmetric-value faulty processor does not become "stuck at the right value." That is, it is possible for a symmetric-value faulty processor to fail in such a way that it appears good. To be reliably detected, a symmetric-value faulty processor must manifest its fault by failing on a value other than the correct value.

Now we go back to the definitions of *Correctness_Prop*, *Correctness*, *Completeness_Prop*, and *Completeness*. The definitions are essentially just the inductive variants of the corresponding final theorems. The lemmas assert that the inductively defined property is true for any number of rounds of execution. The inductive version of Correctness, *Correctness_Prop* also contains some extra assumptions, such as the fact that if a good processor accuses some other processor, then all good processors accuse that processor. This strong property is needed inductively, even though it is not needed explicitly in the theorem (since it is trivially true of the empty set of accusations that starts the protocol).

The complete theory specification is presented in the Appendix section 12.1. Detailed PVS theories can be obtained from lincoln@csl.sri.com or http://www.csl.sri.com/~lincoln.

4.4 Formal Proofs for PP

The formal verifications corresponding to the two main requirements are proved by induction on the number of rounds, and follow the general proofs quite closely. The theorem prover of PVS with its built-in arithmetic decision procedures and rewriting allowed the formal proof to be constructed at a relatively high level without being mired in detail. The PVS system allows partial proofs to be replayed under alternative assumptions, facilitating the exploration of generalizations and special cases.

The first lemma, *fincard_all*, states that the cardinality of an entire set of processors is equal to the sum of the cardinalities of the processors in that set of each status. This lemma follows from properties implicit in the definition of *statuses*: that they are inclusive and disjoint. In detail, the formal proof requires about twenty user-supplied steps in PVS, about ten of which are `ground` or `assert`, which invoke the ground decision procedures of PVS.

The second lemma, *send5*, states that all non-arbitrary-faulty processors exhibit symmetric sending behavior. The proof of this property appeals to the fact that the three statuses—*symmetric-value faulty*, *benign*, and *good*—are inclusive. Case analysis and appeal to the send axioms *send1*, *send2*, and *send4* essentially completes the proof. In two cases, such as that when the transmitter

is benign faulty, the relevant axiom must be applied twice. The entire formal proof comprises 14 user-supplied steps in the PVS interactive verification system.

The other significantly complicated proof is that of Completeness, which is roughly half the size and complexity of the proof of Correctness. Full machine-readable PVS specifications and PVS proofs of the entire proof chain are available from lincoln@csl.sri.com.

Note: As highlighted in Section 2.4, the formal specification of PP develops the formal infrastructure for the specification of the subsequent algorithms PLP, DD and HD. The formal specifications of these algorithms will re-use the formal concepts developed for PP. This will also lead to simpler discussions for the formal analysis of PLP, DD and HD.

5 Model PLP: Considering Byzantine Faults

Model PP is representative of systems which discount the likelihood of asymmetric dispersal and Byzantine faults. If a diagnosis algorithm based on this model encounters an asymmetric-value fault, two interesting cases result. These cases are encountered in steps D3.1.1 and D3.2.1 of algorithm PP. Define $mess_j$ as an error-free message from j and $mess_j^{'}$ an arbitrarily corrupted message from j.

Case (i): Suppose a majority of monitoring processors receive $mess_j^{'}$ and all others receive $mess_j$. Accordingly, if $F_{tot}^n(j) \geq E$ and a monitoring processor, k, does not receive $mess_j^{'}$, then k will be diagnosed as a faulty monitor. The result is that an asymmetric-value fault causes a non-faulty resource, k, to be excluded and early exhaustion of system resources will occur unnecessarily.

Case (ii): Suppose a majority of monitoring processors receive $mess_j$ and all others receive $mess_j^{'}$. Accordingly, if $F_{tot}^n(j) < E$ and a monitoring processor, k, is a minority observer of $mess_j^{'}$ then it will be concluded that k is faulty since an insufficient number of other monitoring processors will support k's accusation.

The typical response to case (i) would be to halt the diagnosis at this point since the source fault j could still be correctly diagnosed. Case (ii) is usually ignored by other diagnosis algorithms because it is assumed that a latent fault of this type will not be harmful to the system during the mission or will hopefully be detected by an extensive off-line diagnosis algorithm. Both of these approaches, however, have the disadvantage of increasing the probability of system failure due to latent faults. This violates a major goal for highly-critical fault tolerant systems where the accumulation of latent faults is to be avoided so the probability of multiple faults can be decreased.

In order to properly resolve these cases, the model of the system is extended to consider the interactions among processors and communications links and their contribution to the cause of asymmetric and symmetric errors in communication. The system structure provides us with some important information. First, a processor cannot maliciously send different messages because of the broadcast mode used by a transmitter. An asymmetric dispersal may be due to: a weak transmitter, environmental noise corrupting messages, or physical damage. Although it is possible to identify errors with greater resolution using the existence of "I'm alive" messages from a transmitting processor, all of these errors will be classified in model PLP as link errors.

The assumption A1 in the original system model will now be modified to give the *processor-link-processor* (PLP) model:

MA1.1 A processor issues a single message onto its associated communications link.

MA1.2 A faulty link may arbitrarily corrupt delivery of the message but cannot correct an erroneous message by its associated processor.

Definition The set of *symmetric* errors, $FS_i^n(j)$, are those messages which i believes that j issued during round n which should be observed by all non-faulty monitoring processors.

Definition The set of *asymmetric* errors, $FA_i^n(j)$, are those messages which i believes that j issued during round n which may be observed only by a partial set of non-faulty monitoring processors.

The primary example of asymmetric dispersal would be a transmission error potentially attributable to a faulty link j. This information would be classified as an *improperly formatted message*, $IFM_i^n(j)$, and be included in $FA_i^n(j)$. It should be noted that detection of this type of error does not necessarily guarantee that all processors made different observations, but that congruent observations cannot be guaranteed. This fact can be used to determine whether it is possible for a distributed system to make a correct diagnosis of the fault based on current syndrome information.

Algorithm PLP

We define $IFM_i^n(j)$ as the set of $mess_j \in \mathcal{M}_i^n(j)$ which failed to pass the link protocol and parity checks. The syndrome information is subdivided into two parts to identify observations regarding symmetric and asymmetric information dispersal. The error report $F_i^n(j)$ now consists of an ordered quintuple $< i, j, n, FS_i^n(j), FA_i^n(j) >$.

D1.0 Monitor each $mess_j \in \mathcal{M}_i^n(j)$.

- **D1.1** If $mess_j \in IFM_i^n(j)$, then $mess_j \in FA_i^n(j)$,
- **D1.2** If $mess_j \in ILM_i^n(j)$, then $mess_j \in FS_i^n(j)$.
- **D1.3** If $mess_j \in MM_i^n(j)$, then $mess_j \in FS_i^n(j)$.

D2. At the completion of round n, for every j, each i will determine if an error report should be issued: if $FA_i^n(j) = FS_i^n(j) = \emptyset$, then do not send a report on j, else send $F_i^n(j)$ as sourced by i to other processors.

D3. For each j, as round $n+1$ completes, compute $F_{tot}^n(j)$.

- **D3.1** If $F_{tot}^n(j) \geq \lceil N/2 \rceil$ then declare j as faulty.
  - **D3.1.1** If a majority of reports agree that only symmetric errors were observed and processor k failed to report any errors then $mess_k \in FS_i^{n+1}(k)$
- **D3.2** If $F_{tot}^n(j) < \lceil N/2 \rceil$ then
  - **D3.2.1** If processor k issued a report on j without identifying an asymmetric observation, then $mess_k \in FS_i^{n+1}(k)$

D4. Increment round counter n and proceed to step D1.

5.1 Analysis of PLP

Algorithm PLP considers the effects introduced in diagnosing errors whenever an asymmetric link error occurs. We will first show that algorithm PLP preserves the property of correctness.

Theorem 5 *(Correctness) If i is non-faulty, then $F_i^n(j) \neq \emptyset$ implies that j is faulty, ensuring the correctness of diagnosis of Algorithm PLP.*

Proof: The proof of Theorem 1 can be extended by including the case where link errors exist. A faulty link can only cause a message to be received improperly or to become lost. The additional set of errors caused by a message failing the link protocol and parity checks is detectable by the receiver as $mess_j \in IFM_i^n$. Thus, if $F_i^n(j) \neq \emptyset$, it will be because j is faulty. □

Before demonstrating that algorithm PLP has the property of completeness, several lemmas will first be introduced.

Lemma 3 *If $\forall\ i \in N$, $FS_i^n(j) \neq \emptyset$ and $FA_i^n(j) = \emptyset$ then processor j is faulty.*

Proof: In order for $FA_i^n(j) = \emptyset$, all links must be non-faulty. The results for model PP now apply as shown in the proof of the correctness property in Theorem 1.

Lemma 4 *If a link, ℓ_j, is faulty in round n, this does not necessarily imply that for each $i \in N$ that $F_i^n(j) \neq \emptyset$.*

Proof: Let ℓ_j be faulty and i non-faulty. If $F_i^n(j) \neq \emptyset$, then it is because i observed an error. However, ℓ_j can arbitrarily fail so that a set of processors, P_{good}, receive correct copies and another set of processors P_{fail}, receive corrupted copies. Thus, it can be seen that not all $i \in N$ observe the error and if i does not detect j to be faulty in any way, $F_i^n(j) = \emptyset$. □

Lemma 5 *For $N \geq 3$, the number of non-faulty processors, i, required to detect an arbitrary failure in processor j is $P_{min} = max(2, t_{mon} + 1)$.*

Proof: We first note that because the algorithm performs diagnosis on each j separately, there is only one source fault j to be dealt with at a time and that j is prevented from participating in its own diagnosis. Thus, we only need to analyze the processors accusing j and their ability to make a correct decision. At least two non-faulty processors need to report $F_i^n(j) \neq \emptyset$ to assure that an invalid alarm does not cause a non-faulty j to be diagnosed as faulty. Now, suppose j is arbitrarily faulty and a set of processors, P_{fail}, record $F_i^n(j) \neq \emptyset$. Even though j's errors may

be detected differently, each i in P_{fail} can be used as a vote indicating j is faulty. Although a majority of non-faulty monitoring processors, i.e. $t_{mon}+1$, are still required, the nature of the asymmetry does not need to be agreed upon but only whether j committed an error. With this fact, if $|P_{fail}| \geq P_{min} = max(2, t_{mon}+1)$, a correct decision can be reached even in the presence of arbitrarily faulty diagnostic processors. □

As with all other diagnosis algorithms, the ability of algorithm PLP to satisfy the completeness property for an arbitrarily faulty j is conditional on the error occurring with enough visibility so that correct diagnosis can be guaranteed.

Theorem 6 *(Completeness) If j is faulty, then at least $P_{min} = max(2, t_{mon}+1)$ non-faulty processors will declare j as faulty, processors, ensuring completeness of diagnosis for Algorithm PLP.*

Proof: Lemma 4 requires that $F^n_{tot}(j)$ non-faulty processors must observe an error by an arbitrarily faulty j, where $F^n_{tot}(j) \geq P_{min} = max(2, t_{mon}+1)$. The rest of the proof is identical to Theorem 2 in that if j sends an erroneous $mess_j \in \mathcal{M}^n_i(j)$, a non-faulty i will always detect an error and report $F^n_i(j) \neq \emptyset$. Also if j omits a $mess_j$ from $\mathcal{M}^n_i(j)$, it will be detectable by a non-faulty i and result in $F^n_i(j) \neq \emptyset$. □

Next, we will analyze its ability to detect failures in the diagnostic processors by determining whether an alarm was valid or invalid. As earlier, we will assume a system contains six processors which are denoted $p_1,\ p_2,\ ...,\ p_6$ and their associated links are $\ell_1, \ell_2,\ ..., \ell_6$ respectively. Let p_1 be faulty during round n. If p_1 is faulty and ℓ_1 is non-faulty, then by Lemma 2 we know that $p_2,\ p_3,\ p_4,\ p_5,\ p_6$ should all respond with $FS^n_i(1) \neq \emptyset$ and $FA^n_i(1) = \emptyset$ since it will be agreed that observations should be congruent. Any processor k which fails to report the error will be accused as faulty such that $FS^{n+1}_i(k) \neq \emptyset$.

Now let p_1 be non-faulty and ℓ_1 faulty. Lemma 3 demonstrates that $p_2,\ p_3,\ p_4,\ p_5,\ p_6$ may not necessarily respond with $F^n_i(1) \neq \emptyset$. If $P_{min} = max(2, t_{mon}+1)$ and $F^n_{tot}(j) < P_{min}$, then j (i.e., p_1 in our example) will not be declared as faulty. Any $F^n_k(j) \neq \emptyset$ will be viewed as an invalid alarm. The only legitimate reason for a non-faulty processor to issue a report on j would be that an error was observed caused by asymmetric information dispersal.

Thus, if there were asymmetric errors (either benign or value faulty) included in the report on j, processor k may have been the sole observer of j's fault. It cannot be determined if k or j is faulty; therefore, no decision is made at this time (step 3.2.1). If only symmetric errors were reported (either benign or value faulty), then k is invalid since these errors should have been observed by at least P_{min} non-faulty processors.

If $F^n_{tot}(j) \geq P_{min}$, then j (or p_1 in the example) will be declared as faulty. Any $F^n_i(j) \neq \emptyset$ will be a valid alarm. The case of diagnosing missing alarms is shown in step D3.1.1. The only legitimate reason for a non-faulty processors failing to respond to a faulty j would be that j sent a correct message to some nodes and an erroneous message to others. If at least P_{min} processors report an error, i.e., $FA^n_k(j) \neq \emptyset$ then j is faulty but monitoring processors still cannot be accused of failing to report an error on j.

6 Formal Analysis for Algorithm PLP

As in the case of PP and F-PP, we first develop a simplification of algorithm PLP as originally presented above [35]. The relationship between this version, F-PLP, and the original PLP is the same as the relationship between PP and F-PP as presented earlier in Section 3.

Algorithm F-PLP:

1. For each i, monitor each message $mess_j$ sent by j to i.
2. If $mess_j$ is improperly formatted, is missing, or properly formatted but fails reasonableness checks, then i accuses j of an asymmetric error.
3. All processors exchange their sets of accusations.
4. Each processor i sums the accusations against every other processor j.
5. If the majority of processors accuse j of some kind of fault, i declares j faulty. In this case, if a majority of processors accuse j of symmetric errors, but some processor k does not accuse j, then i adds a new accusation against k for the next round.
6. If the majority of processors do not accuse j of some kind of fault, and some processor k accuses j of a symmetric fault, then i adds a new accusation against k for the next round.

6.1 Discussion

The main difference between PLP and PP is that PLP differentiates between two kinds of apparently bad values (i.e., locally detectable): those that are produced symmetrically and those that are produced asymmetrically. That is, benign faults produce one kind of error value and link faults produce a different kind of error value.

Since this distinction cannot be guaranteed, some processor may still fail in such a way that it sends out errors on some channels that appear to the receivers to have been sent by a benign-faulty processor at the same time that this failed processor sends good or IFM to some other processors. Such circumstances appear to be extremely rare, but can be disastrous: algorithm PLP is not able to withstand even a single arbitrary fault. Consider the case of a two-round algorithm PLP, with four processors participating, three good and one arbitrary-faulty. The arbitrary-faulty processor, i, sends symmetric-appearing data to one receiver k and good data to the other two receivers. That is, an erroneous value that appears to have come from a benign-faulty processor is sent to one processor k. In any case, the unfortunate processor k will send out an accusation that i is benign-faulty. Since there will be only a single accusation against k, in step 6 the other two good processors Will therefore accuse i of being benignly bad in the next round. With the arbitrary-faulty processor concurring, in the next round all processors other than i itself will accuse i of a symmetric-benign fault, and therefore all other processors will declare i bad. This process could repeat itself with the arbitrary-faulty processor k singling out a single good processor at each round, and quickly exhausting the system resources.

This rare but devastating possibility is a large part of the motivation for the development of the later algorithms algorithms DD and HD (see Section 8). However, for the purposes of the analysis

here, the fault-effect of link faults is restricted to eliminate the possibility described above, and truly arbitrary faults are essentially eliminated from consideration during the analysis of PLP.

The assumptions regarding restricted behavior of links during algorithm PLP is reminiscent of reasonable restrictions one might place on the behavior of interstages. That is, one might assume that an interstage, with only message forwarding capability, is not able to forge signatures nor deliver multiple corrupt messages. The distinction between benign faults and link faults is still difficult, however. A series of cut wires could directly lead to a series of missing messages exactly as described in the scenario above.

In any event, with this restriction of fault-effects, the PLP algorithm above is formalized similar to algorithm PP.

6.2 Requirements and Assumptions

The requirements and assumptions here are the same as for algorithm PP, with the added possibility of link faults. We refer the reader to the Appendix section 12.2 for the formal specification of PLP. As we commented in Section 2.4, once the basic formal specification and analysis is established for an algorithm, (e.g., Sec. 4 for PP), analysis of subsequent algorithms that utilize similar underpinnings can be considerably simplified. This is chiefly due to the element of re-usability of formal concepts entailed in the development of axioms and formal theories which remain essentially unchanged.

7 Developing Foundations for Diagnosis under the Hybrid Fault-Effects Model (HFM)

As discussed in the earlier sections, especially Section 2, our interest is in considering diagnosis of faults with unrestricted behavior. In the previous sections we have described the formal basis for identification of the faulty units. In this section, we continue the discussion of Section 2.2.1 generalizing the underlying fault model to include a composite time and data-domain fault model. We adopt the Hybrid Fault-Effects Model (HFM)[32, 36], in which faults are classified according to the fault manifestations they present across the system. Further, to address the on-line diagnosis process we will base the diagnosis on the existing system consensus procedures. A two-fold motivation is provided for using HFM as linked to the diagnosis objective.

First, we are basing diagnosis on consensus algorithms. These algorithms are very robust but also expensive (time and space complexity) to implement. Designed to provide coverage to the worst case Byzantine faults, these assume all fault occurrences to be Byzantine faults and require a node-redundancy of $N > 3t$ to cover t faults where each instance of t is treated as a Byzantine fault. Realistically, even if the fault was a simple benign fault that could be handled through voting, the traditional approaches require the consensus algorithm to execute. This is both unnecessary and also expensive in terms of time and overhead requirements.

The HFM assumes perfect coverage to a limited number of arbitrary faults, but recognizes that weaker fault types are typically more common than the classical Byzantine faults. The algorithms under the HFM do not compromise the system's capability of tolerating Byzantine faults, however

they provide additional and concurrent coverage to *fault sets (or combinations)* of weaker manifestations. In essence, the same consensus algorithm is shown to have a greater capability of handling a combination of faults of varying severity. This also facilitates a higher resolution of fault granularity compared to considering all faults of a single fault severity. The distinction here is that hybrid faults can be of any type, as long as they can be tolerated by the system implementation *without causing system failure*. If the system is destroyed, or a sufficiently large portion of the system is damaged, then the issue of diagnosis becomes moot.

Second, it is important to note that the fault classes are *disjoint*, under the HFM, preventing any ambiguities in discerning the fault behavior and effect and subsequent diagnostic ambiguities. Furthermore, as the HFM considers fault classification based on the *effect* the fault causes to the system operation, it provides a uniform framework to handle both time-domain and data-domain faults.

7.1 Does consensus hold under the HFM?

Our diagnosis protocols utilize the Consensus paradigm or its extension, Exact Agreement, as relevant, for the fault model under consideration. For the classical fault models we used for algorithms PP and PLP, the classical agreement protocols also applied. The HFM represents a unique fault model which covers a set of faults instead of discrete fault types in earlier models. Thus, prior to developing the diagnostic procedure under HFM, we first need to establish that exact agreement is indeed possible under the HFM. This is not a trivial problem. Algorithm $Z(r)$[13] (see [34]) was proposed to accomplish consensus under the HFM, but was found to be flawed [13]. A corrected algorithm $OMH(r)$ has been formally verified using PVS [13]. To be precise, we review algorithm OMH(r) and its properties below. Although this is a slight digression from the diagnosis issues under consideration here, this is an essential basis for the subsequent diagnosis algorithms DD and HD that follow in Sections 8 and 9.

It is essential to highlight that whereas classical off-line techniques generally assume the sanity of diagnosis operations in an off-line fault-free scenario, on-line techniques such as ours must also consider faults in the diagnostic hardware during the diagnosis process. Thus, we need to ensure that the consensus algorithms such as OMH(r), which form the basis of our diagnostic processes, are also resilient to faults under the HFM. Given the lack of such concepts, we believe the development of OMH(r) under the HFM to be a contribution in itself, and thus the following discussion in this section.

7.1.1 OMH(r) Algorithm for Exact Agreement under HFM

The *Oral Messages Algorithm* [11] demonstrated that Agreement can be guaranteed if $N > 3t$ and $r \geq t$ where t is the number of Byzantine faults, N is the number of nodes, and r is the number of rebroadcast rounds. However, it makes the pessimistic assumption that all faults are asymmetric-value faulty. This implies that a 4, 5, or 6 node system can tolerate only a single arbitrary fault (t=1), and that 7 nodes will be required to tolerate two faults (t=2). However, their analysis is pessimistic in the case of simple modes of failure.

[13] "r" is the number of rounds of message exchange.

Table 2: OMH(r) Algorithm

S1: The Transmitter sends its personal value, v, to all receivers.

S2: $\forall i$, let v_i denote the value that Receiver i gets from the Transmitter.

If $r = 0$, and a benign-faulty**(b)** value is received, Receiver i adopts $\mathcal{E}$. Otherwise, Receiver i adopts v_i. The algorithm then terminates.

If $r > 0$, each Receiver adopts $R(\mathcal{E})$, if a benign-faulty**(b)** value is received, and $R(v_i)$ otherwise. Each receiver then acts as the Transmitter in Algorithm OMH($r-1$) sending its adopted value to the other $N-2$ nodes.

S3: $\forall i, j$, with $i \neq j$, let v_j denote the value Receiver i gets from sender j in Step 2 of OMH($r-1$). If no message is received or v_j is obviously incorrect, Receiver i adopts $\mathcal{E}$ for v_j; otherwise, v_j is used.

Since all Receivers act as senders in OMH($r-1$), each Receiver will have a vector containing (N-1) values at the end of OMH($r-1$). Receiver i adopts v=$H - maj(v_1, v_2, \ldots v_{N-1})$ as the Transmitter's value.

The OMH(r) algorithm — Table 2 — is a r-rebroadcast round protocol based on the OM[11] algorithm, requiring $N > 2a+2s+b+r$ nodes to mask $(a+s+b)$ faults, where b nodes are benign faulty, s nodes are symmetric-value faulty, a nodes are asymmetric-value faulty, and $a \leq r$. Any consensus algorithm must provide the following two key properties under the HFM if there are enough non-faulty nodes and enough rounds.

Validity: If the Transmitter is *non-faulty*, then all non-faulty Receivers select the sender's original value. If the Transmitter is *benign faulty*, then all non-faulty Receivers will adopt a default value, $\mathcal{E}$. If the Transmitter is *symmetric-value faulty*, then all non-faulty Receivers will adopt the value sent by the Transmitter.

Agreement: All non-faulty Receivers agree on the value of the Transmitter.

OMH(r) uses a family of error values, $\{\mathcal{E}, R(\mathcal{E}), \ldots, R^r(\mathcal{E})\}$, where r is the number of rebroadcast rounds. If the value $R^n(\mathcal{E})$ is received, where $n \geq r - l$ in S2 of OMH($r-l$), then that too is recognized as an error, and $\mathcal{E}$ should be adopted. The function H-*maj*, used by OMH(r), is as follows. Given a set V of n values, $v_i, \ldots, v_n$, H-$maj(V)$ is given by

$$\text{H} - maj((V)) = \begin{cases} \mathcal{E}, & \text{if all of the } v_i \text{ satisfy } v_i = \mathcal{E}. \\ R^{-1}(v_{\mathcal{E}}), & \text{if } v_{\mathcal{E}} = maj\,(exclude(V, \mathcal{E})) \text{ exists.} \\ v_0, & \text{otherwise.} \end{cases}$$

The provision which adopts $\mathcal{E}$ if all the v_i are $\mathcal{E}$ cannot occur on a good node, and thus is not required in order for OMH(r) to satisfy the Validity and Agreement properties, but is included to provide a fail safe default value should that case occur on a partially faulty node.

7.2 Algorithm OMH(r): Properties

We defer detailed discussion of the formal analysis of algorithm OMH(r) referring the interested reader to [13]. Below we review the properties relevant to the formalization of the diagnosis procedures.

Lemma 6 *Algorithm OMH(r) achieves Validity for any $a, b, s, N, r \geq 0$, such that $N > 2a + 2s + b + r$, and $a \leq r$.*

Lemma 7 *If $N > 2a + 2s + b + r$, for any $N, a, s, b \geq 0$, $r > 0$ and $a \leq r$, then OMH(r) satisfies Agreement.*

Taken together, Lemmas 6 and 7 prove the following theorem.

Theorem 7 *Algorithm OMH(r) achieves Byzantine Agreement when $N > 2a + 2s + b + r$, for any $r \geq 0$, any $a \leq r$, any $s \geq 0$, and any $b \geq 0$, where a is the number of asymmetric-value faults, s is the number of symmetric-value faults, and b is the number of benign faults in the system.*

This is a basic result which underlies the diagnosis algorithms. However, the diagnosis algorithms (and most applications of consensus in digital flight control) actually require a variant of consensus called Interactive Consistency. Interactive Consistency differs from Consensus in that each processor is assumed to have a value, and the desired result is the reliable interchange of all values so that all good processors agree on all values and if a processor behaves correctly the correct value is the agreed upon value. The obvious solution to this problem is to iterate the OMH(r) algorithm where each processor takes its turn acting as the transmitter. In fact, the HIC problem appears as the inductive case of algorithm OMH(r), as explicitly revealed in [2].

Algorithm 1 *(HIC(r)) Let S be the set of nodes holding values upon which HIC is desired, with $|S| = N$. Each node sends its private value to all other nodes in S, acting as the transmitter in OMH(r), with the value of r identical for all nodes.*

At the conclusion, each good node in S will hold a final vector which satisfies the following conditions.

HIC Validity: Each element of the final vector that corresponds to a non-faulty node is the private value of that node. Each element of the final vector that corresponds to a benign faulty node is $\mathcal{E}$.

HIC Agreement: All non-faulty nodes compute the same vector of values.

Theorem 8 *Algorithm HIC(r) achieves Validity and Agreement when $N > 2a + 2s + b + r$, for any $r \geq 0$, any $a \leq r$, any $s \geq 0$, and any $b \geq 0$, where a is the number of asymmetric-value faults, s is the number of symmetric-value faults, and b is the number of benign faults in the system.*

Proof: By Theorem 7, at the conclusion of OMH(r) with node i as the transmitter, Validity and Agreement will be satisfied. Thus, each good node will have a consistent view of node i's personal value. All nodes then execute OMH(r). By the proof of agreement and validity for OMH, if $N > 2a + 2s + b + r$, and all nodes act as the Transmitter in OMH(r), with $a \leq r$, and all nodes using the same value of r, then Validity and Agreement will be satisfied. Since i is arbitrary, this guarantees that all good nodes will hold the same vector of values, either a good node's original value or an agreed upon default value for a faulty node's original value. □

The combination of the OMH(r) and the HIC(r) algorithms provide a formal basis to our diagnostic approach presented in Sections 8 and 9.

7.3 Formal Analysis for OMH(r)

For this analysis, we build on the formal analyses developed in earlier work on Byzantine Agreement and interactive consistency under a hybrid fault model [13, 13]. We use the same fault model with the exception of ruling out link faults. Thus any link fault is counted as an arbitrary fault, since it may not present symmetric data to all receivers.

The goal of OMH(r) is to distribute single-source data (such as a sensor sample) from one processor of a fault-tolerant system to all the others in such a way that all non-faulty processors receive the same value. The principal difficulty is that a faulty processor may send different values to different receivers. To overcome this, traditional algorithms use several "rounds" of message exchange during which processor p tells processor q what value it received from processor r and so on. Under the "Oral Messages" assumptions, the difficulty is compounded because a faulty processor q may "lie" to processor r about the value it received from processor p [11]. We assume there are n processors in total, one of which is distinguished as the transmitter (or the "General", in more informal terms).

The problem is to devise an algorithm that will allow each receiver p to compute an estimate ν_p of the transmitter's value satisfying the following requirements:

- **Agreement**: If processors p and q are non-faulty, then they agree on the value ascribed to the transmitter; that is, $\nu_p = \nu_q$.
- **Validity**: If processor p is non-faulty, and the transmitter is not arbitrary-faulty, the value ascribed to the transmitter by p is the value actually sent from v to p.

The basic design of OMH(r) is that the transmitter first sends a value to all other processors. Each receiver then plays the part of the transmitter in a recursive instance of the algorithm. Each receiver then takes a vote of the values it has received and uses the majority value as its final value. However, in each round of OMH(r), the processors do not forward the actual value they received. Instead, each processor sends a value corresponding to the statement "I'm reporting *value*." If a benign faulty value is received, it is recorded as the special value E. After several rounds, it is possible to imagine values corresponding to "I'm reporting that he's reporting that she's reporting E" arise. When taking the majority vote, processors ignore all E values, but treat "I'm reporting E" values as regular values. After the majority vote, if the result is "He is reporting Y," then "Y" is taken as the final value. The algorithm OMH(r) is defined semi-formally below. The parameter m is the number of rounds of message exchanges that are to be performed; the functions R and UnR correspond to the addition and removal of the "I'm reporting" tags and are specified in more detail shortly.

OMH(0)

1. The transmitter sends its value to every receiver.
2. Each receiver uses the value received from the transmitter, or uses the value E if a missing or benign-faulty value is received.

OMH(r), r > 0

1. The transmitter sends its value to every receiver.
2. For each p, let v_p be the value receiver p obtains from the transmitter, or E if no value, or a benign faulty bad value, is received.

   Each receiver p acts as the transmitter in Algorithm OMH($m-1$) to communicate the value $R(v_p)$ to all of the $N-1$ receivers, including itself.
3. For each p and q, let v_q be the value receiver p received from receiver q in Step (2) using Algorithm OMH($r-1$), or else E if no such value, or a benign-faulty value, was received. Each receiver p computes the majority of all non-E values v_q received (if no majority exists, the receiver uses some arbitrary, but functionally determined value), and then applies UnR to that value, using the result as the transmitter's value.

7.4 Sketching out the Formal Proof

For this algorithm to be correct as stated, we must make three more assumptions: (1) The class of possible messages exchanged between processors can be increased to accommodate new kinds of messages such as "I'm reporting that she's reporting 5." (2) For all values v, $R(v) \neq E$; reported errors are never mistaken for errors. (3) For all values v, $UnR(R(v)) = v$; untagging a tagged value results in the original value. The addition of the tagging and untagging functions, R and UnR, is key to the correctness of Algorithm OMH(r).

The argument for the correctness of OMH is an adaptation of that for the Byzantine Generals formulation

- N, the number of processors
- a, the maximum number of arbitrary-faulty processors
- s, the maximum number of symmetric-value faulty processors
- b, the maximum number of benign-faulty processors
- r the number of rounds of message passing the algorithm is to perform.

Theorem 9 *For any r, Algorithm OMH(r) satisfies validity if there are more than $2(a+s)+b+r$ processors.*

Proof: This is proved by induction on r. In the base case, the assumptions on message transmission ensure the property. For the inductive case, in Step 2 of the algorithm all non-faulty receivers apply the algorithm OMH($r-1$) to the value received. By a counting argument, we apply the inductive hypothesis to conclude that non-faulty receivers correctly record the forwarded value. All values forwarded from non-faulty processors are of the form $R(x)$ for some value x. By another counting argument, we see that the non-faulty processors form a majority of the processors which are not benign-faulty, and therefore the values $R(x)$ forwarded by them win the majority vote, and after applying UnR, all non-faulty receivers settle on the value actually sent by the transmitter. □

Theorem 10 *For any r, Algorithm OMH(r) satisfies agreement if there are more than $2(a+s)+b+r$ processors and $r \geq a$.*

Proof: This theorem is also proved by induction on r. In the base case, there can be no arbitrary faulty processors, since $r \geq a$ and $r = 0$; by the previous theorem, therefore, we have the result. In the inductive step there are two cases: (1) when the transmitter is arbitrary-faulty, and (2) otherwise. In the latter case, again the previous theorem is sufficient. When the transmitter is arbitrary faulty, the inductive hypothesis can be applied by a counting argument to show that all non-faulty processors arrive at the same set of values before the hybrid majority vote is taken. If there is a majority, all non-faulty processors will agree on that value; if there is no majority, all non-faulty processors will agree on the functionally determined value. Whatever that value, all non-faulty processors will arrive at consistent values after applying the function *UnR*. □

We utilize OMH(r) to distribute locally diagnosed syndrome information in algorithm DD, discussed in Section 8, and in HD and F-HD, a formal version of algorithm HD, discussed in Section 9. Appendix section 12.3 outlines the formal specifications for the OMH(r) algorithm.

8 On-Line Diagnosis Under the HFM: Algorithm DD

Diagnosis consists of two parts: *(1)* errors must be properly detected, and *(2)* correctly classified according to their behavior. In the earlier algorithms PP and PLP, we demonstrated the feasibility of correctly detecting errors of arbitrary behavior during on-line operations. We now shift our focus to the classification process and further detail those aspects of the diagnosis algorithm.

As the system is a frame-based model, we utilize these frame delimiters to define diagnosis intervals, in which the following primitives are executed: *local detection and diagnosis*, *global information collection* and *global diagnosis* – on a concurrent, on-line and continual basis. The information collected locally by each node during diagnosis interval n, $\mathcal{D}(n)$, is broadcast to all other nodes, which then collect and analyze the information during $D(n+1)$, to formulate a global perspective on the fault behavior. The length of the diagnosis interval is bounded by the assumed frequency of asymmetric-value faults in the system.

We continue our presentation of diagnosis algorithms on a progressive basis. Algorithm DD (*Distributed Diagnosis*) represents the basic two-phase, on-line diagnostic approach linking the diagnosis and consensus procedures. We will utilize algorithm DD to introduce the diagnostic rationale under the HFM, and present the details and formal analysis for the subsequent algorithm HD. In Algorithm HD (*Hybrid Diagnosis*), we extend on DD to provide the capability of discriminating between node and link faults, where possible; to assess the severity of the node fault from temporal perspectives, and to incorporate the facets of node recovery and re-integration.

8.1 Distributed Diagnosis: Node Health Monitoring

The goal of algorithm DD, shown in Table 3, is for all non-faulty nodes to acquire a consistent view of the health of all other nodes in the system. This is done by an exchange of local health assessments and subsequent computation of a consistent global health vector, using the OMH(r) algorithm. It

Table 3: Algorithm DD for Node i : Basic Consensus Driven Syndrome Formulation

DD0 Initialize s_i^n to be the zero health vector.

DD1 [*Local detection:*] Over round $\mathcal{D}(n)$, $\forall i, j \in N$, each processor i monitors each message $mess_j$ $\forall j \neq i$.

DD1.1 If received message is $mess_j'$[a], update $\sigma_{ij}^n = (\sigma_{ij}^n$ OR 1), thus forming the local n^{th} round health vector s_i^n.

DD2 For each i, j, as round $D(n)$ completes:

DD2.1 [*Global Dispersion:*] Broadcast updated health vector $s_i^n =< \sigma_{i1}^n, \sigma_{i2}^n, \ldots \sigma_{iN}^n >$.

DD2.2 [*Global Assimilation:*] Node i collects health vectors s_i^{n-1} computed during diagnosis interval $\mathcal{D}(n-1)$ into the $N \times N$ global syndrome matrix $S_i^{(n-1)}$. Row l of $S_i^{(n-1)}$ is the syndrome vector $s_l^{(n-1)}$ received from node l.

DD3 [*Syndrome Matrix/Diagnosis:*] Combine the values in column j of $S_i^{(n-1)}$, corresponding to other nodes' views of the health of node j, by using a hybrid voting function to generate a consistent health value, faulty or non-faulty, for node j. Node exclusion/inclusion as per consensus value exceeding $\lceil \frac{N}{2} \rceil$.

DD4 Increment round counter n and proceed to step DD0.

[a] Unlike PP and PLP segregation of ILM, MM and IFM classes, $mess_j'$ encompasses all faults locally discernible by processor i

is pertinent to mention that the diagnosis algorithm running during diagnosis interval $\mathcal{D}(k)$ utilizes information collected across the system over the previous round $\mathcal{D}(k-1)$. All operations can be considered to be on-going in a pipelined manner. It may be noted that the basic operations of PP and PLP are integral to the operation of DD.

Inter-node messages are considered as the sole indicators of the health of a node. Based on this, step DD1, the local detection phase, examines all received messages for errors. Since the detection of an error in a message by its receiver implies that the sender is *locally benign faulty*, local detection utilizes the parity checks, checksums, message framing checks, range checks, sanity checks, and comparison techniques. The failure to receive an expected message from a node or an early/delayed message is also logged as an error for that node.

During $\mathcal{D}(n)$, each node i locally formulates a health vector, $s_i^n =< \sigma_{i1}^n, \sigma_{i2}^n, \ldots, \sigma_{iN}^n >$, containing an entry, σ_{ij}^n, corresponding to the perceived status of each system node, j. If any error is detected from a given node, j, its entry σ_{ij}^n is set to 1; otherwise it remains at the fault-free value of 0. This local diagnosis step is equivalent to the identity mapping, as no further local diagnosis occurs following detection.

To achieve consistency of this local diagnosis, and to build the global perspective to handle the "locally undetectable" class of *value* faults, information dispersal across the system is necessitated. At the end of each detection interval, in step DD2.1, the local health vector s_i^n of node i is sent to all other nodes. Thus, each node compiles (and analyzes) a global syndrome matrix during $\mathcal{D}(n+1)$ that contains the local health assessments of every node by other nodes over $\mathcal{D}(n)$.

During $\mathcal{D}(n)$, global diagnosis of each node's health during $\mathcal{D}(n-1)$ is performed in step DD3. The local health vectors computed during $\mathcal{D}(n-1)$ form the rows of the global health matrix $S_i^{(n-1)}$. If no health vector or an discernibly erroneous vector is received from node l, then an error indicator value, $\mathcal{E}$, is adopted for each $\sigma_{ij}^{(n-1)}$ in $s_l^{(n-1)}$, and node l is assessed for an error

by updating σ_{ij}^{n}. The global health vector held by each non-faulty node i is denoted by $h_i^{(n-1)}$, with entries $\eta_{ij}^{(n-1)}$ giving the global status of node j during $\mathcal{D}(n-1)$. The global health vector for $\mathcal{D}(n-1)$ is computed during $\mathcal{D}(n)$ by applying a hybrid majority voting function, described below, to each column of $S_i^{(n-1)}$.

First, all elements of column j equal to $\mathcal{E}$ are excluded, along with node j's opinion of itself ($\sigma_{jj}^{(n-1)}$). The final value, $\eta_{ij}^{(n-1)}$, is the majority of the remaining values. If no majority exists, the value 0 should be adopted to ensure that a good node is not identified as faulty. At the conclusion of $\mathcal{D}(n)$, each good node will contain a global health vector $h_i^{(n-1)} =< \eta_{i1}^{(n-1)}, \eta_{i2}^{(n-1)}, \ldots, \eta_{iN}^{(n-1)} >$, where $\eta_{ij}^{(n-1)} = 1$ means that node i has diagnosed node j as being faulty during $\mathcal{D}(n-1)$. It needs to be stated that this diagnosis is consistently achieved by all non-faulty nodes, as this is the HIC-Agreement condition.

8.2 Formal Requirements for DD

The requirements for algorithm DD to provide for diagnosis are as follows:

- **Correctness**: If there are more than $2a + 2s + b + 1$ processors, $a \leq 1$, and DD diagnoses j as faulty, then j is indeed faulty.
- **Completeness**: If there are more than $2a + 2s + b + 1$ processors, $a \leq 1$, and some processor j is benign-faulty, then DD will diagnose j as faulty.

Algorithm DD is thus required to diagnose simple faults in the presence of arbitrary faults. As discussed before, it is impossible to reliably diagnose all arbitrary faults. Thus, here the requirement is only to diagnose the benign-faulty processors in the presence of arbitrary-faulty processors.

There is no requirement for DD to diagnose either benign or value faults. In fact, as pointed out in an earlier paper [36], algorithm DD is incomplete with respect to arbitrary faults. For example, consider the case where there are four total processors, one arbitrary-faulty and three good. The arbitrary-faulty processor may send apparently bad[14] messages to one of the good processors. Even though that processor will report the faulty message, there are not enough accusations to allow the other good processors to determine which processor is faulty.

We present the complete formal specification of algorithm DD in the Appendix section 12.4.

9 Algorithm HD: Hybrid Fault Diagnosis

In DD, an error in a single message from a node during a single diagnosis interval is sufficient to cause that node to be removed from the system. The local detection mechanism described previously for DD1 treats a node that sends a single erroneous message as if all messages from the node were indeed faulty. Essentially, a transient and a permanent fault will have an identical fault effect here. This is not an efficient strategy, and could lead to rapid depletion of system resources. Further, DD provides only the fault detection and isolation facets of FDIR. Recovery of a faulty

[14] locally detectable

Table 4: Fault Classification under HD

Type		Recorded in:
$\mathcal{MM}_i^n(j)$	Missing Message	ϵ_{ij1}^n
$\mathcal{IFM}_i^n(j)$	Improperly Formatted Message	ϵ_{ij2}^n
$\mathcal{ILM}_i^n(j)$	Improper Logical Message	ϵ_{ij3}^n
$\mathcal{CVM}_i^n(j)$	Failing Comparison to Voted Value	ϵ_{ij4}^n

node or node re-integration require refined temporal considerations, which the DD does not fully support. These issues, and increasing the diagnostic resolution, are dealt with in HD (Table 5), building on the basic framework of DD. We also add temporal fault detection to the local diagnosis primitive, and replace the simple good–bad local diagnosis with a preliminary assessment of the node and/or link fault symmetry.

9.1 Local Primitives in HD

The scalar status value, σ_{ij}^n, used in step DD1 of the previous algorithm, corresponding to node i's local assessment of node j's health during $\mathcal{D}(n)$, is replaced in step HD1 by a local diagnosis vector, e_{ij}^n. This local diagnosis vector is used to indicate the type of detection mechanism which found errors in messages from node j, providing a preliminary diagnosis of the fault type. While the entries in $e_{ij}^n =< \epsilon_{ij1}^n, \epsilon_{ij2}^n, \ldots, \epsilon_{ijm}^n >$ can have a one-to-one correspondence with the m fault detection mechanisms implemented in the system, as in MAFT [37, 35], we consider four potential diagnoses shown in Table 4, as refined over those presented in Sec. 8.

The information in the local error log, e_{ij}^n, is condensed into a set of boolean error flags and a cumulative penalty weight which is sent in the summary error report on an accused node j. The boolean flags indicate that one or more errors of a given type were observed. An individual flag can represent either a symmetric or asymmetric communication error and flag definitions are determined by a small lookup table which is programmed identically in each processor. One example of the flexibility provided by this table is that the model can be selected by programming all flags to be symmetric (model PP) or identifying the specific asymmetric flags (model PLP/HD). The use of boolean error flags reduces the total amount of information to be transmitted for reporting an error; however, multiple occurrences of the same error type cannot be represented. The cumulative penalty weight performs this function and reflects the detection of all errors. Also, by aggregating all errors over round n, the size and total number of reports can be fixed.

We assign a penalty weight to each error type, commensurate with its assumed severity in the system implementation, and accumulate the weights for each node over $\mathcal{D}(n)$. By definition, these detected errors result from *benign* faulty nodes. However, discerning the potential symmetry of the errors is useful in discriminating between a crash faulty node and a potentially less severe faulty communications link. The relationships among these accumulated penalty counts, referred to as ω_{MM}, ω_{ILM}, ω_{IFM}, ω_{CVM} for the MM, ILM, IFM and CVM error categories, forms the basis of inferences on fault-type. Of course, more or fewer weights could be used. Also, an additional correlated weight can be used if a faulty node exhibits several of these behaviors during a single diagnosis interval. It is interesting to note that selection of different weights to different fault types is very much application dependent. Essentially, for different system operation profiles this

Table 5: Algorithm HD for node i : Enhanced Consensus Driven Syndrome Formulation

HD0 Initialize expanded health vector, $s_i^n =< e_{i1}^n, e_{i2}^n, \ldots, e_{iN}^n >$, to zero at the beginning of $\mathcal{D}(n)$; restore penalty weight vector $p_i^n =< \rho_{i1}^n, \rho_{i2}^n, \ldots, \rho_{iN}^n >$ from D(n-1).

HD1 [*Local Diagnosis:*] Over round $\mathcal{D}(n)$, $\forall i, j \in N$, each processor i monitors each message $mess_j$ $\forall j \neq i$.

- **HD1.1** If $mess_j \in \mathcal{MM}$, then set $\epsilon_{ij1}^n = \max(\epsilon_{ij}^n, 1)$ and $\rho_{ij}^n = \rho_{ij}^n + \omega_{MM}$.
- **HD1.2** If $mess_j \in \mathcal{IFM}$, then set $\epsilon_{ij2}^n = \max(\epsilon_{ij2}^n, 1)$ and $\rho_{ij}^n = \rho_{ij}^n + \omega_{ILM}$.
- **HD1.3** If $mess_j \in \mathcal{ILM}$, then set $\epsilon_{ij3}^n = \max(\epsilon_{ij3}^n, 1)$ and $\rho_{ij}^n = \rho_{ij}^n + \omega_{IFM}$.
- **HD1.4** If $mess_j \in \mathcal{CVM}$, then set $\epsilon_{ij4}^n = \max(\epsilon_{ij}^n, 1)$, and $\rho_{ij}^n = \rho_{ij}^n + \omega_{CVM}$.

HD2 For each i, j, as round $D(n)$ completes:

- **HD2.1** [*Global Dispersion:*] Broadcast updated health vector, $s_i^n =< \sigma_{i1}^n, \sigma_{i2}^n, \ldots \sigma_{iN}^n >$, and penalty count vector, $p_i^n =< \rho_{i1}^n, \rho_{i2}^n, \ldots, \rho_{iN}^n >$.
- **HD2.2** [*Information Assimilation:*] Node i collects the health vectors computed during $\mathcal{D}(n-1)$ into the $N \times N$ syndrome matrix $S_i^{(n-1)}$, where row l of $S_i^{(n-1)}$ is the syndrome vector $s_l^{(n-1)}$ received from node l. Similarly, the penalty counts computed during $\mathcal{D}(n-1)$ are collected into the matrix $P_i^{(n-1)}$, where column j of $P_i^{(n-1)}$ contains the weights received from all nodes regarding node j.

HD3 [*Consensus* $\rightarrow$ *Diagnosis:*] Combine the values in column j of $S_i^{(n-1)}$ to generate a health value, faulty or good, for node j by first OR-*ing* the entries in $e_{ij}^{(n-1)}$, and then performing a hybrid majority vote down the column. Combine the values in column j of $P_i^{(n-1)}$ to generate a consistent incremental penalty count for node j during $\mathcal{D}(n-1)$ using a hybrid voting function. Node Inclusion/Exclusion based on threshold matching to Incremental and Cumulative Penalty Counts.

HD4 Increment round counter n and proceed to step HD0.

provides an effective way of controlling the system features of availability metrics and also defining the fail-safe/stop conditions as desired or warranted for the particular application profile. The final extended health vectors and accumulated penalty weights from $\mathcal{D}(n)$ are sent to all nodes at the end of $\mathcal{D}(n)$.

9.2 HD: Global Diagnosis/Properties

During $\mathcal{D}(n)$, the extended health vectors and penalty weights from all nodes during $\mathcal{D}(n-1)$ are analyzed in a fashion similar to that in DD. Steps HD2 and HD3 ensure a consistent global perspective on the cumulative penalty values associated with each and every system node following global information collection in HD2. The overall relative value of a fault type e.g., $\epsilon_{ij1} > \epsilon_{ij2}$'s, are useful in attempting to identify the type of fault. The penalty weight for each node under diagnosis is its initial value in OMH(1). The asymmetric fault coverage is limited to one fault by the single re-broadcast round.

Since behavior of a faulty node is unrestricted, and a faulty node can send different corrupt health vectors (or none) to other nodes, good nodes may receive different health matrices. So, we must prove that the final health vectors h_i computed by all good nodes i during each diagnosis interval are consistent. Furthermore, we must assess the correctness and completeness of diagnosis. Global diagnosis is *correct* for $\mathcal{D}(n-1)$ if each node identified as faulty by a good node in step

HD3 of HD (during $\mathcal{D}(n)$) is indeed faulty. Similarly, global diagnosis is *complete* for $\mathcal{D}(n-1)$ if all nodes that were faulty during $\mathcal{D}(n-1)$ are identified as such in step HD3 in that diagnosis interval.

While the statement of algorithm *HD* does not explicitly invoke any fault tolerance algorithm, it is implicit in the definition of HD and in the fault masking used in the the **HIC(a)** algorithm. The local health of node j as viewed by node i during $\mathcal{D}(n-1)$, assessed as either 0 or 1, is equivalent to node j holding a personal value of either 0 or 1 and transmitting it to node i during $\mathcal{D}(n-1)$. This corresponds to the initial round of the **OMH(a)**, with $r = 1$. The sending by node i of its local assessment of node j's health to other nodes at the end of $\mathcal{D}(n-1)$ represents the rebroadcast round of OMH(1), with the hybrid majority column vote of $S_i^{(n)}$ during interval $\mathcal{D}(n)$ equivalent to the final value for j as computed by node i in OMH(1).

Since *HD* is executed on all nodes, and each node i monitors all other nodes, completion of *HD* is equivalent to all good nodes achieving interactive consistency during $\mathcal{D}(n)$ on the health of all other nodes during $\mathcal{D}(n-1)$. Theorems 7 and 8 provide conditions under which the global health vectors h_i^{n-1} are guaranteed to be consistent on all good nodes. These theorems also permit us to prove the following correctness and completeness results.

Theorem 11 *(*Correctness*) If $n > 2t - b + 1$, where $t = a + b + s$ faulty nodes are present during both $\mathcal{D}(n-1)$ and $\mathcal{D}(n)$, and $a \leq 1$, then diagnosis under* Algorithm HD *during $\mathcal{D}(n)$ is guaranteed to be correct for $\mathcal{D}(n-1)$.*

Proof: By Theorems 7 and 2, with the default majority value set to 0, all good nodes will have $h_{il}^{(n-1)} = 0$ for the values of all good nodes l. Thus, any node j for which $h_{ij}^{(n-1)} = 1$ must be faulty. □

Theorem 12 *(*Completeness*) If $n > 2t - b + 1$ as defined above, $a \leq 1$, and node j was benign faulty during $\mathcal{D}(n-1)$, then under* Algorithm HD*, node j will be diagnosed during $\mathcal{D}(n)$ as having been faulty during $\mathcal{D}(n-1)$.*

Proof: By definition, a node that is benign faulty during $\mathcal{D}(n-1)$ will be detected locally on each good node. Thus, the sender's initial value for these nodes is 1, with all good nodes adopting 1 (faulty) as the local diagnosis during $\mathcal{D}(n-1)$, and agreeing on 1 at the conclusion of $\mathcal{D}(n)$, by Thm 7. □

This covers the cases for data and node faults. It must however be kept in mind that correctness and completeness conditions for value faults (symmetric or asymmetric) is ensured after the re-broadcast round of health vectors is completed.

10 Formal Analysis for Algorithm HD

Next we present a re-write of algorithm HD, termed as F-HD, for the formal analysis. As for F-PP and F-PLP, the purpose of the simplification, F-HD, is to reduce the notational complexity and emphasize the operations. The main simplification between F-HD and HD are that F-HD uses the current rounds data to diagnose faults, F-HD makes more explicit the exchange of some computed values, and F-HD exchanges only the penalty weights, where HD broadcasts both penalty weight

and boolean health information. In F-HD the assumption is that a node i accuses another node j of being faulty if and only if it broadcasts a non-zero penalty weight. One could combine algorithm DD and F-HD to provide both boolean health information (DD) and penalty weight information (F-HD) into a combined algorithm that would more closely simulate the above algorithm HD. As was done for algorithms F-PP and F-PLP, developing the formal arguments quite often helps to better understand the algorithms themselves, and also be able to provide clarifications or modifications for better working of the algorithms.

10.1 Algorithm F-HD

Each processor sends its report s_i^k to all other processors.

F-HD(0)

1. All accusations of faults are cleared by setting p_i^k to be the zero penalty vector.

F-HD(k), k > 0

1. Each processor i executes one frame of the workload, arriving at some value or checksum $Val(i)$
2. Each processor sends $Val(i)$ to all other processors.
3. Initialize $p_i^k = \rho_{i1}^k, \rho_{i2}^k, \cdots, \rho_{in}^k$ to be the zero health vector
4. Node i monitors message traffic from all other nodes throughout F-HD(k). If the value received from j is apparently bad, set $\rho_{ij}^k = 1$ in s_i^k.
5. Node i reliably distributes its health vector $p_i^k = \rho_{i1}^k, \rho_{i2}^k, \cdots, \rho_{in}^k$ to all other nodes using OMH(1).
6. Node i collects health vectors p_l^k into the $n \times n$ global health syndrome matrix $P_i^{(K)}$. Row l of $P_i^{(K)}$ is the syndrome vector $p_l^{(k)}$ received from node l in the previous step.
7. Node i combines the values in each column j of $P_i^{(K)}$ using a hybrid majority voting function to generate a consistent health value, faulty or non-faulty, for node j. If the penalty value is nonzero, i declares j faulty.

As stated above, we have not described detailed formal analysis of algorithm HD. Instead we have developed a variant called F-HD. Algorithm HD is essentially algorithm DD with some extra mechanisms to separate fault-effects and to perform fault severity estimation as well as fault decays. As with algorithm DD, algorithm HD and HD's agreement component are analogous to OMH(1), which provides resilience against only a single arbitrary fault.

10.2 Formal Specification for HD

The formal specification of algorithm F-HD follows the specification of DD in Section 8 and 12.4 in most respects. One key difference is the instance of the OMH(r) theory imported. The specification of algorithm DD is parametric in the types of values being voted on. DD passes the type of values being voted on to the OMH(r) theory, since agreement (algorithm OMH(r)) will be run on these values. On the other hand, the specification of algorithm F-HD assumes that the values being voted are natural numbers. This choice is made in order to simplify the specification slightly as well as to demonstrate that it is possible to instantiate the OMH(r) specification with a concrete type (natural numbers). However, even this instance is still abstract in that it presents no bound on the size of natural numbers used. As has been argued elsewhere [13], an effective bound on the size of natural numbers can be maintained.

The instance of OMH(r) utilized in the specification of F-HD below uses algorithm OMH(r) to allow processors to agree on a natural number. The other parameters to the OMH(r) theory are the number of rounds used, a specific error value, and two functions called `act` and `unact` in OMH(r) theory. These functions are implemented over the naturals as the increment and decrement functions. However, the decrement function must ensure that it always returns an natural number, even in the case its input is zero, and is thus called *Decrement*.

One other difference between the specification of algorithms F-HD and DD is that algorithm DD is working over the boolean domain, and thus the output value can be used directly in the statement of theorems, while F-HD is working with natural numbers. Thus a top level function *HDtop* is defined as a simple threshold predicate over the natural number results of *HD*.

The detailed formal specification of F-HD is presented in the Appendix Section 12.5.

10.3 Algorithm HD: Addressing Temporal Perspectives

Following the formalism of the HD algorithm, we now motivate the fault resolution and the temporal aspect of aggregating the penalty weights in steps HD1 and HD3. It should be kept in mind that the processor and link fault diagnosis of PLP is directly encompassed under HD.

Algorithm HD improves the judgment of fault severity at any interval in time so that units with less severe fault indications are left operational. Additional processing is required in HD as we are interested in handling fault-effects over a longer round of time than the diagnostic interval. We can build on the results obtained by instantaneous diagnosis(DD) by placing them into a temporal framework(HD).

Errors can be viewed as the manifestation of faults which exist in the system. *Duration* is defined as the total time a fault and its effects are present in the system during actual operation. We can introduce the concept of *decay-time* to be the length of time an error would be present if the fault was instantaneously applied and removed. Thus, the error is the effect of an instantaneous fault injected at time t_0 which lasts for a time Δt_f. The concept of *decay-time* allows error information to be carried across multiple intervals so that instantaneous diagnosis information can be related over time.

It must be noted that the *decay-time* does not always correlate directly with the severity level of the error. If a function in the system hard core is impacted by the transient fault, it may be

necessary to immediately deal with the effects rather than waiting for them to die out. This method can account for the possibility that errors may propagate due to lack of containment and cause other errors which have their own *decay-time* and severity.

Errors which have shorter *decay-times* will have less time to further impact system operation. For example, a lost bit on a communication link due to a transient fault should be considered as an error with a short *decay-time*. If a noise pulse affects the link, some time will need to pass before the energy is dissipated from the medium. During this time, the messages being sent may be corrupted, depending on the level of noise. Another example would be a memory module with scrubbing. When an an error occurs, there will be a time period where the error could propagate and induce further errors. Once the scrubbing mechanism detects the error and removes it, the immediate danger of error propagation will have lapsed (even though the faulty source may still be present and/or intermittent).

Decay rates can therefore be determined if there exist regular and predictable times where errors can be detected or removed. If the times of detection are not regular, it is prudent to assume a worst case scenario which can be arrived at in a number of ways. The first approach would be to assess a penalty so severe it causes exclusion immediately so that further reliance on fault detection is not needed. A second method would be to attempt to identify the worst case detection time by a higher level mechanism. This method may allow for some error propagation until it begins to affect a critical higher level function. A third alternative is to schedule more extensive FDIR tasks to attempt to collect more information while imposing greater overhead on the system.

Based on the concept of *decay-time*, we can assume that if the fault is applied and then goes away immediately, the fault-effects should only last for a certain period of time. Faults with their associated *decay-times* are handled as follows. First, a penalty weight (W_e) should be assessed against the faulty unit in the interval $\mathcal{D}$(k). Second, during the following diagnostic intervals, for the node displaying sustained fault-free behavior, the penalty weight against it is reduced by a predetermined amount, referred to as the *decrement-count* (DC). The ratio, W_e/DC provides the *decay-time* for the given error.

This *decrement-count* is introduced so that temporary malfunctions do not result in permanent exclusion. Note, that if the fault persists, penalties will continue to accrue and the decrement amount will be offset by new increases in penalty weights. The duration aspect of faults is also handled in this model. For the fault being transient, the source of the fault is removed and the effects should disappear after the appropriate decay time has passed. For a permanent fault, the source of the fault will remain present and new penalties will continually accrue over each new diagnostic interval until an exclusion threshold is reached. In order to handle transient and intermittent faults which are severe enough to cause exclusion but allow re-admission, the exclusion and re-admission thresholds must be appropriately separated.

This approach alternately supports modeling of permanent faults by setting the decrement count to zero. If one wants to support graceful re-admission of system units with on-line repair, even permanent faults can have a relatively short *decay-time*. The decay time can also be based on the time of re-admission, since once the unit is repaired its count must be decremented for it not to appear faulty anymore.

11 Conclusion

Most existing diagnosis strategies treat diagnosis as a stand-alone process in the system operations, and are primarily off-line techniques. In this paper, we have addressed the problem of performing on-line diagnosis as an integral phase of the system FDIR process. Unlike existing approaches, the strategy is based on monitoring the system message traffic rather than using explicit test procedures. We believe our work to be the first to present an on-line diagnosis solution incorporated into the FDIR approach for a distributed environment.

Extending beyond the fixed fault severity models (time-domain and data-domain, s-a-X, Byzantine faults), the HFM framework is developed, which permits handling a continuum of fault types as groups of faults of varying fault manifestations under a single algorithm. The HFM's applicability to diagnosis is formally shown through the development of a distributed agreement algorithm (OMH) and diagnosis algorithms DD and HD. The integration of HFM into the diagnosis domain facilitates increased diagnostic resolution which can be used for improved resource management strategies.

The overall approach presented in this paper is unique in presenting a comprehensive diagnostic solution to span the entire range of encountered faults from the simplest benign fault case to the extreme arbitrary fault cases, all within a single diagnostic paradigm, and as an on-line solution. We have developed algorithms which provide for diagnostic support for progressively more comprehensive and generalized fault models. Furthermore, we have presented diagnostic algorithms which, unlike the simplifying assumption in classical approaches of diagnosis being conducted in a fault-free off-line environment, not only provide on-line diagnosis, but are also tolerant to faults in the diagnostic process.

Furthermore, we have presented techniques of formal analysis and verification which greatly help to rigorize the correctness and completeness of the developed algorithms, and also help develop insights into the operations of the algorithms. Such an approach is shown to result in better understanding of the algorithms and for providing clarifications, and sometimes, modifications for better working of the algorithms using a systematic and rigorous formal specification and verification approach.

The paper has also introduced the concept of fault *decay-time* and its impact in handling transient, intermittent and permanent faults in conjunction with Byzantine faults. The usage of penalty counts is shown as a basis for graceful node exclusion and re-admission protocols. In future work a detailed penalty count model will be developed. Overall we have shown that a more efficient and flexible approach to FDIR can be constructed, supported by on-line diagnosis algorithms under a generalized hybrid fault-effects model.

Acknowledgments: We would like to acknowledge the contributions of M.M Hugue to parts of Sections 7, 8 and 9. We extend a very sincere appreciation for a very thorough and diligent review process from the reviewers. They have very significantly helped in raising the quality of the paper.

References

[1] Barborak, M. *et al.*, "The consensus problem in fault-tolerant computing," *ACM Computing surveys*, vol. 25, pp. 171–220, June 1993.

[2] Bevier, W. and Young, W., "Machine-checked proofs of a Byzantine agreement algorithm", *TR 55*, CLI, Austin, TX, June 1990.

[3] Boyer, R. and Moore, J., "MJRTY—a fast majority vote algorithm," In Robert S. Boyer, volume 1 of *Automated Reasoning Series*, pp. 105–117. Kluwer, 1991.

[4] Dolev, D. and Strong, H., "Authenticated algorithms for Byzantine algorithms," *SIAM Journal of Computing*, vol. 12, pp. 656–666, Nov 1983.

[5] Gong, L., Lincoln, P. and Rushby, J., "Byzantine agreement with authentication: Observations and applications in tolerating hybrid and link faults", *DCCA-5*, 1995.

[6] Henke, F. *et al.* The EHDM verification environment: An overview. In *Proceedings 11th National Computer Security Conference*, pages 147–155, Oct. 1988.

[7] Holzmann, G., "Design and Validation of Computer Protocols", *Prentice-Hall*, 1991.

[8] Kieckhafer, R. *et al*, "The MAFT architecture for distributed fault tolerance", *IEEE Transactions on Computers*, 37(4):398–405, April 1988.

[9] Kuhl, J. and Reddy, S., "Distributed fault-tolerance for large multiprocessor systems", In *7th Intl. Symposium on Computer Architectures*, pp. 23–30, 1980.

[10] Kuhl, J. and Reddy, S., "Fault-diagnosis in fully distributed systems", In *FTCS-11*, pp. 100–105, 1981.

[11] Lamport, L. *et al.*, "The Byzantine generals problem," *ACM Trans. on Prog. Languages and Systems*, vol. 4, pp. 382–401, July 1982.

[12] L. Lamport and P. M. Melliar-Smith, "Synchronizing clocks in the presence of faults", *Journal of the ACM*, 32(1):52–78, Jan. 1985.

[13] Lincoln, P. and Rushby, J., "A formally verified algorithm for interactive consistency under a hybrid fault model," *FTCS-23*, pp. 402–411, 1993.

[14] Lincoln, P. and Rushby, J., "Formal verification of an interactive consistency algorithm for the Draper FTP architecture under a hybrid fault model", In *COMPASS '94*, pp. 107–120, 1994.

[15] Mallela, S. and Masson, G., "Diagnosis without repair for hybrid fault situations", *IEEE Transactions on Computers*, C-29:461–470, June 1980.

[16] Meyer, F. and Pradhan, D., "Consensus with dual failure modes", *IEEE Transactions on Parallel and Distributed Systems*, 2(2):214–222, April 1991.

[17] Owre, S. *et al*, "Formal verification of fault tolerant architectures - Prolegomena to the Design of PVS", *IEEE Transactions on Software Engg.*, pp. 107–125, vol 21, # 2., Feb 1995.

[18] Owre, S. *et al*, "The PVS specification language", TR CSL-SRI, Feb. 1993.

[19] Owre, S. *et al*, "User guide to PVS specification and verification system," TR CSL-SRI, Feb. 1993.

[20] Pease, M. Shostak, R. and Lamport, L., "Reaching agreement in the presence of faults", *Journal of the ACM*, 27(2):228–234, April 1980.

[21] Preparata, F., Metze, G. and Chien, R., "On the connection assignment problem of diagnosable systems", *IEEE Transactions on Computers*, EC-16(6):848–854, December 1967.

[22] Ramarao, K. and Adams, J., "On the diagnosis of Byzantine faults", In *7th Symposium on Reliable Distributed Systems*, pp. 144–153, 1988.

[23] Rushby, J., "Reconfiguration and transient recovery in state m/c architectures", *FTCS-26*, pp. 6-25, 1996.

[24] Rushby, J., "Formal verification of an Oral Messages algorithm for interactive consistency", TR SRI-CSL-92-1, CSL, SRI 1992.

[25] Rushby, J., "A formally verified algorithm for clock synchronization under a hybrid fault model", In *Thirteenth ACM Symposium on Principles of Distributed Computing*, pp. 304–313, 1994.

[26] Rushby, J. and Henke, F., "Formal verification of the Interactive Convergence clock synchronization algorithm using EHDM", TR SRI-CSL-89-3R, CSL, SRI, 1991.

[27] Rushby, J. and Henke, F., "Formal verification of algorithms for critical systems", *IEEE Transactions on Software Engineering*, vol. 19, pp. 13–23, Jan. 1993.

[28] Sengupta, A. and Dahbura, A., "On self-diagnosable multiprocessor systems: diagnosis by the comparison approach", *IEEE Transactions on Computers*, C-41:1386–1396, Nov. 1992.

[29] Sengupta, A. *et al*, "On system diagnosability in the presence of hybrid faults", *IEEE Transactions on Computers*, C-35:90–93, Jan. 1986.

[30] Shankar, N., "Mechanical verification of a generalized protocol for Byzantine fault-tolerant clock synchronization", In J. Vytopil, editor, *Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 571 of *Lecture Notes in Computer Science*, pages 217–236, 1992.

[31] Shin, K. and Ramanathan, P., "Diagnosis of processors with Byzantine faults in a distributed computing system," *FTCS-17*, pp. 55–60, 1987.

[32] Suri, N. *et al.*, "Reliability modeling of large fault-tolerant systems," *FTCS-22*, pp. 212–220, 1992.

[33] Suri, N., Walter, C. and Hugue, M., "Advances in Ultra-Dependable Distributed Systems," *IEEE CS Press*, 1995.

[34] Thambidurai, P. and Park, Y., "Interactive consistency with multiple failure modes," *Proc. of RDS*, pp. 93–100, 1988.

[35] Walter, C., "Identifying the cause of detected errors," *FTCS-20*, pp. 48–55, June 1990.

[36] Walter, C., Suri, N. and Hugue, M., "Continual on-line diagnosis of Hybrid Faults," *DCCA-4*, Jan 1994.

[37] Walter, C. J. *et al.*, "MAFT: A multicomputer architecture for fault-tolerance in real-time control systems," *RTSS*, 1985.

[38] Yang, C. and Masson, G., "Hybrid fault diagnosability with unreliable communication links", In *FTCS-16*, pp. 226–231, 1986.

[39] Yang, C. and Masson, G., "A new measure for hybrid fault diagnosability", In *IEEE Transactions on Computers*, Vol. C-36, No. 3, pp. 378-382, 1987.

[40] Young, W., "Verifying the Interactive Convergence clock-synchronization algorithm using the Boyer-Moore prover", *NASA TR 189649*, NASA LARC, April 1992.

12 Appendix

The PVS theories of all presented algorithms are available from lincoln@csl.sri.com and also at http://www.csl.sri.com/~lincoln.

12.1 Formal PP Specification

Finally, here we give the entire theory specification:

pp[m : posnat, n : posnat, T : TYPE, error : T, BAD : $\{x : T \mid \neg\, x = \text{error}\}$,
GOOD : $\{x : T \mid (\neg\, x = \text{error}) \wedge (\neg\, x = \text{BAD})\}$,
Val : [upto[m] $\rightarrow$ $\{x : T \mid \neg\, (x = \text{error} \vee x = \text{BAD} \vee x = \text{GOOD})\}$]] :
THEORY
BEGIN

rounds : TYPE = upto[m]

t : VAR T

fcu : TYPE = below[n]

fcuset : TYPE = setof[fcu]

fcuvector : TYPE = [fcu $\rightarrow$ T]

G, p, q, z : VAR fcu

v, v_1, v_2 : VAR fcuvector

caucus : VAR fcuset

r, R, R_2 : VAR rounds

PSET : TYPE = fcu

pset : VAR setof[PSET]

i, j, k, i_2, j_2 : VAR PSET

Accuse, OldAccuse : VAR [PSET, PSET $\rightarrow$ bool]

AllDeclare : VAR [PSET, PSET $\rightarrow$ bool]

IMPORTING card_set[fcu, n, identity[fcu]],
finite_cardinality[fcu, n, identity[fcu]],
filters[fcu], hybridmjrty[T, n, error]

statuses : TYPE = {symmetric, manifest, good}

status : [fcu $\rightarrow$ statuses]

$g(z)$: bool = good?(status(z))

$s(z)$: bool = symmetric?(status(z))

$c(z)$: bool = manifest?(status(z))

cs(caucus) : fcuset = filter(caucus, c)

ss(caucus) : fcuset = filter(caucus, s)

gs(caucus) : fcuset = filter(caucus, g)

fincard_all : LEMMA
fincard(caucus)
= fincard(cs(caucus)) + fincard(ss(caucus)) + fincard(gs(caucus))

send : [T, fcu, fcu $\rightarrow$ T]

send1 : AXIOM $g(p) \supset \text{send}(t, p, q) = t$

send2 : AXIOM $c(p) \supset \text{send}(t, p, q) = \text{error}$

send4 : AXIOM $s(p) \supset \text{send}(t, p, q) = \text{send}(t, p, z)$

send5 : LEMMA $\text{send}(t, p, q) = \text{send}(t, p, z)$

HybridMajority(caucus, v) : T = PROJ_1(Hybrid_mjrty(caucus, v, n))

HybridMajority1 : LEMMA
fincard(gs(caucus)) > fincard(ss(caucus))
$\wedge$ ($\forall$ p : $g(p) \wedge p \in$ caucus $\supset v(p) = t$)
$\wedge$ $t \neq$ error
$\wedge$
($\forall$ p :
$c(p) \wedge p \in$ caucus
$\supset v(p) =$ error)

```
      ⊃ HybridMajority(caucus, v) = t

  HybridMajority2 : LEMMA
    (∀ p : p ∈ caucus ⊃ v1(p) = v2(p))
      ⊃ HybridMajority(caucus, v1) = HybridMajority(caucus, v2)

  HybridMajority3 : LEMMA
    HybridMajority(caucus, v) = t
      ∧
      (∀ p, q :
        g(p) ∧ g(q) ∧ p ∈ caucus ∧ q ∈ caucus
          ⊃ (v(p) = v(q) ∧ v(p) ≠ error))
        ∧ fincard(gs(caucus)) > fincard(ss(caucus))
            ∧
            (∀ p :
              c(p) ∧ p ∈ caucus
                ⊃ v(p) = error)
      ⊃ (∀ p : g(p) ∧ p ∈ caucus ⊃ v(p) = t)

  Syndrome(R, j, i, OldAccuse) : T =
    IF OldAccuse(i, j) ∨ (¬ Val(R) = send(Val(R), j, i)) THEN BAD
     ELSE GOOD
     ENDIF

  KDeclareJ(pset, R, OldAccuse, j, k) : bool =
    HybridMajority(pset, λ i : send(Syndrome(R, j, i, OldAccuse), i, k))
      = BAD

  PP(pset, R, OldAccuse)(i, j) :
    RECURSIVE bool = IF R = 0 THEN FALSE
      ELSE KDeclareJ(pset, R, OldAccuse, j, i)
        ∨
        PP(pset, R − 1,
           (λ i2, k :
             OldAccuse(i2, k)
               ∨ ∃ j2 :
                   (KDeclareJ(pset, R, OldAccuse, j2, i2)
                      ≠
                     (send(Syndrome(R, j2, k, OldAccuse),
                           k, i2)
= BAD))))(i,
           j)
      ENDIF
      MEASURE (λ pset, R, OldAccuse → nat : R)

  Correctness_Prop(R) : bool =
    (∀ i, j, pset, OldAccuse :
      g(i)
        ∧ i ∈ pset
            ∧ j ∈ pset
                ∧ fincard(gs(pset)) > fincard(ss(pset)) + 1
```

```
                    ∧ PP(pset, R, OldAccuse)(i, j)
                        ∧
                        (∀ p, q, k :
                          ((g(p)
                              ∧ g(q) ∧ OldAccuse(p, k))
                             ⊃ OldAccuse(q, k)
                                 ∧ (c(k) ∨ s(k))))
        ⊃ c(j) ∨ s(j))

Correctness : LEMMA Correctness_Prop(R)

Completeness_Prop(R) : bool =
  (∀ i, j, pset, OldAccuse :
    g(i)
      ∧ i ∈ pset
          ∧ j ∈ pset
              ∧
              (c(j)
                 ∨ (s(j) ∧ (∀ t, p : send(t, j, p) ≠ t)))
                ∧ fincard(gs(pset)) > fincard(ss(pset)) + 1
      ⊃ PP(pset, R, OldAccuse)(i, j))

Completeness : LEMMA (∀ R : Completeness_Prop(R) ∨ R = 0)

Empty(i, j) : bool = FALSE

Final_Correctness : THEOREM
   (∀ i, j :
     g(i)
       ∧ fincard(gs(fullset[fcu])) > fincard(ss(fullset[fcu])) + 1
           ∧ PP(fullset[fcu], R, Empty)(i, j)
       ⊃ c(j) ∨ s(j))

Final_Completeness : THEOREM
   (∀ i, j :
     g(i)
       ∧ (c(j) ∨ (s(j) ∧ (∀ t, p : send(t, j, p) ≠ t)))
           ∧ fincard(gs(fullset[fcu]))
               > fincard(ss(fullset[fcu])) + 1
               ∧ R > 0
       ⊃ PP(fullset[fcu], R, Empty)(i, j))

END pp
```

12.2 Formal PLP Specification

The formal specification of algorithm PLP follows that of algorithm PP quite closely. We point out that the possible fault classes have changed to include link faults, and the algorithm has changed to explicitly check for the two different kinds of error values. The use of hybrid majority now requires

that two different kinds of error be excluded from the vote. However, an implementation where *HybridMajority* excludes only benign faulty values (that is, *Serror*) satisfies all the requirements as well.

Otherwise the specification is essentially unchanged.

```
plp[m : posnat, n : posnat, T : TYPE, Serror : T, Aerror : {x : T | ¬x = Serror},
    BAD : {x : T | (¬x = Serror) ∧ (¬x = Aerror)},
    GOOD : {x : T
     | (¬x = Serror) ∧ (¬x = BAD) ∧ (¬x = Aerror)},
    Val : [upto[m]
             → {x : T
             |
              (¬x = Serror)∧
                (¬x = Aerror)∧
                  (¬x = BAD) ∧ (¬x = GOOD)}]] :
THEORY
 BEGIN

 rounds : TYPE = upto[m]

 t : VAR T

 fcu : TYPE = below[n]

 fcuset : TYPE = setof[fcu]

 fcuvector : TYPE = [fcu → T]

 G, p, q, z : VAR fcu

 v, v1, v2 : VAR fcuvector

 caucus : VAR fcuset

 r, R, R2 : VAR rounds

 PSET : TYPE = fcu

 pset : VAR setof[PSET]

 i, j, k, i2, j2 : VAR PSET

 OldAccuse : VAR [PSET, PSET → bool]

 AllDeclare : VAR [PSET, PSET → bool]

 IMPORTING card_set[fcu, n, identity[fcu]],
           finite_cardinality[fcu, n, identity[fcu]], filters[fcu], hybridmjrty[T, n, Serror]

 statuses : TYPE = {link, manifest, good}
```

status : [fcu → statuses]

$g(z)$: bool = good?(status(z))

$c(z)$: bool = manifest?(status(z))

$l(z)$: bool = link?(status(z))

cs(caucus) : fcuset = filter(caucus, c)

ls(caucus) : fcuset = filter(caucus, l)

gs(caucus) : fcuset = filter(caucus, g)

fincard_all : LEMMA
fincard(caucus) = fincard(cs(caucus)) + fincard(ls(caucus)) + fincard(gs(caucus))

send : $[T, \text{fcu}, \text{fcu} \rightarrow T]$

send1 : AXIOM $g(p) \supset \text{send}(t, p, q) = t$

send2 : AXIOM $c(p) \supset \text{send}(t, p, q) = \text{Serror}$

send4 : AXIOM $l(p) \supset \text{send}(t, p, q) = \text{Aerror} \vee \text{send}(t, p, q) = t$

HybridMajority(caucus, v) : T = PROJ_1(Hybrid_mjrty(caucus, v, n))

HybridMajority1 : LEMMA
fincard(gs(caucus)) > fincard(ls(caucus))∧
($\forall\ p$: $g(p) \wedge (p \in \text{caucus}) \supset v(p) = t$)∧
$t \neq$ Serror∧
($\forall\ p$:
$c(p) \wedge (p \in \text{caucus}) \supset$
$v(p) = \text{Serror}) \supset$
HybridMajority(caucus, v) = t

HybridMajority2 : LEMMA
($\forall\ p$: $(p \in \text{caucus}) \supset v_1(p) = v_2(p)) \supset$
HybridMajority(caucus, v_1) = HybridMajority(caucus, v_2)

HybridMajority3 : LEMMA
HybridMajority(caucus, v) = t∧
($\forall\ p, q$:
$g(p)$∧
$g(q)$∧
$(p \in \text{caucus})$∧
$(q \in \text{caucus}) \supset$
$(v(p) = v(q) \wedge v(p) \neq \text{Serror}))$∧
fincard(gs(caucus)) > fincard(ls(caucus))∧
($\forall\ p$:

```
            c(p) ∧ (p ∈ caucus) ⊃
              v(p) = Serror) ⊃
      (∀ p : g(p) ∧ (p ∈ caucus) ⊃ v(p) = t)

Syndrome(R, j, i, OldAccuse) : T =
  IF OldAccuse(i, j) ∨ (¬Val(R) = send(Val(R), j, i)) THEN BAD ELSE GOOD ENDIF

KDeclareJ(pset, R, OldAccuse, j, k) :
            bool = HybridMajority(pset, λ i : send(Syndrome(R, j, i, OldAccuse), i, k)) = BAD

PLP(pset, R, OldAccuse)(i, j) :
  RECURSIVE bool = IF R = 0 THEN FALSE
    ELSE
     KDeclareJ(pset, R, OldAccuse, j, i)∨
      PLP(pset, R − 1,
        (λ i_2, k :
          OldAccuse(i_2, k)∨
            ∃j_2 :
              (KDeclareJ(pset, R, OldAccuse, j_2, i_2) ≠
                (send(Syndrome(R, j_2, k, OldAccuse), k, i_2) =
                  BAD))))(i,
        j)
    ENDIF
   MEASURE (λ pset, R, OldAccuse → nat : R)

Correctness_Prop(R) : bool =
  (∀ i, j, pset, OldAccuse :
    g(i)∧
      (i ∈ pset)∧
        (j ∈ pset)∧
          fincard(gs(pset)) > fincard(ls(pset)) + 1∧
            PLP(pset, R, OldAccuse)(i, j)∧
              (∀ p, q, k :
                ((g(p)∧
                    g(q)∧
                      OldAccuse(p, k)) ⊃
                  OldAccuse(q, k)∧
                    (c(k) ∨ l(k)))) ⊃
      ¬(g(j)))

Correctness : LEMMA Correctness_Prop(R)

Completeness_Prop(R) : bool =
  (∀ i, j, pset, OldAccuse :
    g(i)∧
      (i ∈ pset)∧
        (j ∈ pset)∧
          (c(j)∨
            (l(j)∧
              (∀ t, p :
                send(t, j, p) ≠
```

```
                t)))∧
            fincard(gs(pset)) > fincard(ls(pset)) + 1 ⊃
        PLP(pset, R, OldAccuse)(i, j))

  Completeness : LEMMA (∀ R : Completeness_Prop(R) ∨ R = 0)

  Empty(i, j) : bool = FALSE

  Final_Correctness : THEOREM
     (∀ i, j :
       g(i)∧
          fincard(gs(fullset[fcu])) > fincard(ls(fullset[fcu])) + 1∧
            PLP(fullset[fcu], R, Empty)(i, j) ⊃
         ¬(g(j)))

  Final_Completeness : THEOREM
     (∀ i, j :
       g(i)∧
          (c(j)∨
             (l(j)∧
               (∀ t, p :
                 send(t, j, p) ≠ t))∧
            fincard(gs(fullset[fcu]))
                  > fincard(ls(fullset[fcu])) + 1∧
              R > 0 ⊃
        PLP(fullset[fcu], R, Empty)(i, j))

  END plp
```

12.3 Formal OMH(r) Specification

A gentle introduction to the formal specification of OMH is omitted from this document. [13] presents a very readable introduction and derivation of this formal specification along with a description of its complete formal verification using PVS [17]. We have used the nomenclature used in [17] for this section.

```
omh[m : nat, n : posnat, T : TYPE, error : T, act, unact : [T -> T]] : THEORY
   BEGIN

   ASSUMING

     act_ax : ASSUMPTION (∀ (t : T) : act(t) ≠ error)

     unact_ax : ASSUMPTION (∀ (t : T) : unact(act(t)) = t)

     ENDASSUMING

   rounds : TYPE = upto[m]
```

t : VAR T

fcu : TYPE = below[n]

fcuset : TYPE = setof[fcu]

fcuvector : TYPE = [fcu $->$ T]

G, p, q, z : VAR fcu

v, v_1, v_2 : VAR fcuvector

caucus : VAR fcuset

r : VAR rounds

IMPORTING card_set[fcu, n, identity[fcu]],
finite_cardinality[fcu, n, identity[fcu]], filters[fcu], hybridmjrty[T, n, error]

statuses : TYPE = {arbitrary, symmetric, manifest, good}

status : [fcu $->$ statuses]

$a(z)$: bool = arbitrary?(status(z))

$s(z)$: bool = symmetric?(status(z))

$c(z)$: bool = manifest?(status(z))

$g(z)$: bool = good?(status(z))

as(caucus) : fcuset = filter(caucus, a)

ss(caucus) : fcuset = filter(caucus, s)

cs(caucus) : fcuset = filter(caucus, c)

gs(caucus) : fcuset = filter(caucus, g)

fincard_all : LEMMA
fincard(caucus)
= fincard(as(caucus)) + fincard(ss(caucus)) + fincard(cs(caucus)) + fincard(gs(caucus))

send : [T, fcu, fcu $->$ T]

send1 : AXIOM $\supset$ $g(p)$send(t, p, q) = t

send2 : AXIOM $\supset$ $c(p)$send(t, p, q) = error

send4 : AXIOM $\supset$ $s(p)$send(t, p, q) = send(t, p, z)

```
send5 : LEMMA ⊃ NOT a(p)send(t, p, q) = send(t, p, z)

HybridMajority(caucus, v) : T = PROJ_1(Hybrid_mjrty(caucus, v, n))

HybridMajority_ax1 : LEMMA
   ⊃ fincard(gs(caucus)) > fincard(as(caucus)) + fincard(ss(caucus))
       AND (∀ p : g(p) AND p ∈ caucus IMPLIES v(p) = t)
        AND t /= error
          AND (∀ p : c(p) AND p ∈ caucus IMPLIES v(p) = error)
    HybridMajority(caucus, v) = t

HybridMajority_ax2 : LEMMA
   ⊃ (∀ p : p ∈ caucus IMPLIES v1(p) = v2(p))
    HybridMajority(caucus, v1) = HybridMajority(caucus, v2)

OMH(G, r, t, caucus) :
 RECURSIVE fcuvector = IF r = 0 THEN (λ p : send(t, G, p))
   ELSE
    (λ p :
      IF p = G THEN send(t, G, p)
       ELSE
        unact(HybridMajority(remove(G, caucus),
                             (λ (q : {s : nat | s < n}) :
                               OMH(q, r − 1, act(send(t, G, q)),
                                   remove(G, caucus))(p))))
       ENDIF)
   ENDIF
  MEASURE (λ G, r, t, caucus −> nat : r)

Validity_Prop(r) : bool =
 (∀ p, q, caucus, t :
   ⊃ NOT a(q)
       AND p ∈ caucus
        AND q ∈ caucus
          AND fincard(caucus)
            > 2 ∗ (fincard(as(caucus)) + fincard(ss(caucus))) + fincard(cs(caucus))
             + r
    OMH(q, r, t, caucus)(p) = send(t, q, p))

Validity : LEMMA Validity_Prop(r)

Agreement_Prop(r) : bool =
 (∀ p, q, z, caucus, t :
   ⊃ (g(p)
       AND g(q)
        AND p ∈ caucus
         AND q ∈ caucus
          AND z ∈ caucus
           AND fincard(caucus)
             > 2 ∗ (fincard(as(caucus)) + fincard(ss(caucus)))
              + fincard(cs(caucus))
```

$+ \ r$
AND $r >=$ fincard(as(caucus)))
OMH$(z, r, t,$ caucus$)(p) = $ OMH$(z, r, t,$ caucus$)(q))$

Agreement : LEMMA Agreement_Prop(r)

Validity_Final : THEOREM
$\supset g(p)$
AND NOT $a(G)$
AND fincard$(a) <= m$ AND $2 * $ fincard$(a) + 2 * $ fincard$(s) + $ fincard$(c) + m < n$
OMH$(G, m, t,$ fullset[fcu]$)(p) = $ send(t, G, p)

Validity_Corollary : THEOREM
$\supset g(p)$
AND $g(G)$
AND fincard$(a) <= m$ AND $2 * $ fincard$(a) + 2 * $ fincard$(s) + $ fincard$(c) + m < n$
OMH$(G, m, t,$ fullset[fcu]$)(p) = t$

Agreement_Final : THEOREM
$\supset g(p)$
AND $g(q)$
AND fincard$(a) <= m$ AND $2 * $ fincard$(a) + 2 * $ fincard$(s) + $ fincard$(c) + m < n$
OMH$(G, m, t,$ fullset[fcu]$)(p) = $ OMH$(G, m, t,$ fullset[fcu]$)(q)$

Crash_Only_Validity_Prop(r) : bool =
$(\forall p, q,$ caucus$, t :$
$\supset g(p)$
AND $p \in$ caucus
AND $q \in$ caucus
AND fincard(as(caucus)) $= 0$
AND fincard(ss(caucus)) $= 0$ AND fincard(caucus) $> r$
OMH$(q, r, t,$ caucus$)(p) = $ send$(t, q, p))$

Crash_Only_Validity : LEMMA Crash_Only_Validity_Prop(r)

Crash_Only_Agreement_Prop(r) : bool =
$(\forall p, q, z,$ caucus$, t :$
$\supset g(p)$
AND $g(q)$
AND $p \in$ caucus
AND $q \in$ caucus
AND $z \in$ caucus
AND fincard(as(caucus)) $= 0$
AND fincard(ss(caucus)) $= 0$ AND fincard(caucus) $> r$
OMH$(z, r, t,$ caucus$)(p) = $ OMH$(z, r, t,$ caucus$)(q))$

Crash_Only_Agreement : LEMMA Crash_Only_Agreement_Prop(r)

Crash_Only_Validity_Final : THEOREM
$\supset g(p)$ AND fincard$(a) = 0$ AND fincard$(s) = 0$ AND fincard(fullset[fcu]) $> m$
OMH$(G, m, t,$ fullset[fcu]$)(p) = $ send(t, G, p)

Crash_Only_Validity_Corollary : THEOREM
$\supset g(p)$ AND $g(G)$ AND fincard$(a) = 0$ AND fincard$(s) = 0$ AND fincard(fullset[fcu]) $> m$
OMH$(G, m, t,$ fullset[fcu]$)(p) = t$

Crash_Only_Agreement_Final : THEOREM
$\supset g(p)$ AND $g(q)$ AND fincard$(a) = 0$ AND fincard$(s) = 0$ AND fincard(fullset[fcu]) $> m$
OMH$(G, m, t,$ fullset[fcu]$)(p) =$ OMH$(G, m, t,$ fullset[fcu]$)(q)$

END omh

12.4 Formal Specifications for DD

The formal specification of algorithm DD follows the specification of PP in Section 3 and OMH in various respects. The key definition of the algorithm DD is simpler in this case than in either previous case, since PP retains *OldAccuse* information from one period to the next, and OMH performs exponentially many recursive message-sends in deeper recursive instances of the algorithm.

The specification of DD does not include the description of the fault model nor of message sending directly. Instead the DD theory imports these specifications from the OMH theory, which is presented in Section 7.1.1.

dd[m : posnat, n : posnat, T : TYPE, error : T, act, unact : [T -> T], BAD : {x : T | NOT x = error},
GOOD : {x : T | NOT $\vee x$ = error$x$ = BAD},
Val : [[upto[m], upto[n], upto[n] -> {x : T | NOT $\vee x$ = error$x$ = BAD OR x = GOOD}]]] :
THEORY
BEGIN

ASSUMING

act_ax : ASSUMPTION $(\forall\ (t :\ T) :$ act$(t) \neq$ error)

unact_ax : ASSUMPTION $(\forall\ (t :\ T) :$ unact(act$(t)) = t)$

ENDASSUMING

ROUNDS : TYPE = upto[m]

R : VAR ROUNDS

t : VAR T

PSET : TYPE = below[n]

PSETvector : TYPE = [PSET -> T]

pset : VAR setof[PSET]

i, j, k : VAR PSET

```
IMPORTING omh[m, n, T, error, act, unact]

Syndrome(R, j, i) : T = IF Val(R, j, i) ≠ send(Val(R, j, i), j, i) THEN BAD ELSE GOOD ENDIF

ICH(pset, (v : PSETvector)) : [PSET -> PSETvector] = (λ i : OMH(i, 1, v(i), pset))

KDeclareJ(pset, R, j, k) : bool = ICH(pset, λ i : Syndrome(R, j, i))(k)(j) = BAD

DD(pset, R)(i, j) :
 RECURSIVE bool = IF R = 0 THEN FALSE
   ELSE ∨ KDeclareJ(pset, R, j, i)DD(pset, R − 1)(i, j)
   ENDIF
  MEASURE (λ pset, R-> nat : R)

Correctness_Prop(R) : bool =
 (∀ i, j, pset :
   ⊃ g(i)
       AND i ∈ pset
         AND j ∈ pset
           AND fincard(pset)
             > 2 * (fincard(as(pset)) + fincard(ss(pset))) + fincard(cs(pset)) + m
             AND fincard(as(pset)) ≤ 1 AND DD(pset, R)(i, j)
    NOT(g(j)))

Correctness : LEMMA Correctness_Prop(R)

Completeness_Prop(R) : bool =
 (∀ i, j, pset :
   ⊃ g(i)
       AND c(j)
         AND i ∈ pset
           AND j ∈ pset
             AND fincard(as(pset)) ≤ 1 AND fincard(pset)
               > 2 * (fincard(as(pset)) + fincard(ss(pset))) + fincard(cs(pset)) + m
    DD(pset, R)(i, j))

Completeness : LEMMA (∀ R : ∨ Completeness_Prop(R)R = 0)

Final_Correctness : THEOREM
   ⊃ g(i)
       AND n > 2 * fincard(a) + 2 * fincard(s) + fincard(c) + m
         AND fincard(a) ≤ 1 AND DD(fullset[PSET], R)(i, j)
    NOTg(j)

Final_Completeness : THEOREM
   ⊃ g(i)
       AND
     c(j)

       AND n > 2 * fincard(a) + 2 * fincard(s) + fincard(c) + m AND fincard(a) ≤ 1 AND R > 0
```

DD(fullset[PSET], R)(i, j)

END dd

12.5 Formal Specification for HD

```
hd2[m : posnat, n : posnat, T : TYPE, Val : [upto[m], upto[n], upto[n] -> posnat],
    decay : nat, manifest_penalty, wrongvalue_penalty : posnat] :
 THEORY
  BEGIN

  ROUNDS : TYPE = upto[m]

  R : VAR ROUNDS

  t : VAR T

  PSET : TYPE = below[n]

  PSETvector : TYPE = [PSET -> T]

  pset : VAR setof[PSET]

  i, j, k : VAR PSET

  Increment((x : nat)) : nat = x + 1

  Decrement((x : nat)) : nat = IF x = 0 THEN 0 ELSE x - 1 ENDIF

  IMPORTING omh[m, n, nat, 0, Increment, Decrement]

  Decay((x : nat)) : nat = x ~ decay

  Penalty(R, j, i) : nat =
   IF send(Val(R, j, i), j, i) = 0 THEN manifest_penalty
    ELSE IF Val(R, j, i) /= send(Val(R, j, i), j, i) THEN wrongvalue_penalty
      ELSE 0
      ENDIF
    ENDIF

  ICH(pset, (v : [PSET -> nat])) : [PSET -> [PSET -> nat]] = (λ i : OMH(i, 1, v(i), pset))

  KPenaltyJ(pset, R, j, k) : nat = ICH(pset, λ i : Penalty(R, j, i))(k)(j)

  F-HD(pset, R)(i, j) :
   RECURSIVE nat = IF R = 0 THEN 0
     ELSE KPenaltyJ(pset, R, j, i) + Decay(F-HD(pset, R - 1)(i, j))
     ENDIF
    MEASURE (λ pset, R-> nat : R)
```

```
F-HDtop(pset, R)(i, j) : bool = F-HD(pset, R)(i, j) > 0

Correctness_Prop(R) : bool =
 (∀ i, j, pset :
    ⊃ g(i)
        AND i ∈ pset
         AND j ∈ pset
          AND fincard(pset)
            > 2 * (fincard(as(pset)) + fincard(ss(pset))) + fincard(cs(pset)) + m
           AND fincard(as(pset)) ≤ 1
           AND F-HDtop(pset, R)(i, j)
     NOT(g(j)))

Correctness : LEMMA Correctness_Prop(R)

Completeness_Prop(R) : bool =
 (∀ i, j, pset :
    ⊃ g(i)
        AND c(j)
         AND i ∈ pset
          AND j ∈ pset
           AND fincard(pset)
            AND fincard(as(pset)) ≤ 1
             > 2 * (fincard(as(pset)) + fincard(ss(pset))) + fincard(cs(pset)) + m
     F-HDtop(pset, R)(i, j))

Completeness : LEMMA (∀ R : ∨ Completeness_Prop(R)R = 0)

Final_Correctness : THEOREM
    ⊃ g(i)
        AND n > 2 * fincard(a) + 2 * fincard(s) + fincard(c) + m
         AND fincard(a) ≤ 1 AND I- HDtop(fullset[PSET], R)(i, j)
     NOTg(j)

Final_Completeness : THEOREM
    ⊃ g(i)
        AND
      c(j)

        AND fincard(a) ≤ 1 AND n > 2 * fincard(a) + 2 * fincard(s) + fincard(c) + m AND R > 0
   F-HDtop(fullset[PSET], R)(i, j)

END hd2
```

13 Biographies

Chris J. Walter received the BSEE degree from the University of Notre Dame in 1975, the MSEE degree from the University of Michigan in 1978, and the D.Sc. degree in computer science and

electrical engineering from the George Washington University in 1989. Dr. Walter has been involved in the field of fault tolerant computing for over 20 years in the areas of aerospace, automotive and telephony. He has authored over 25 journal and conference papers and been awarded 13 US patents and 9 international patents. He has co-authored an IEEE tutorial text on Readings in Ultra-dependable Distributed Systems in 1995. His work has also been included in other textbooks in fault tolerant computing systems. In April 1994, he served as chairman for the International Workshop on Integrating Error Models with Fault Injection.

Prior to founding WW Technology Group in 1994, Dr. Walter was head of the computer research and systems integration group at AlliedSignal for the company's aerospace divisions. He was responsible for technology in the areas of fault tolerant computing, distributed systems, and parallel processing. Some of his projects of interest include the design of a fly-by-wire flight control system for the Boeing 7J7 (precursor to the 777) and an advanced engine controller architecture for the Space Shuttle Main Engines. Included among his current projects is the development of a scalable fault tolerant system architecture for real-time applications.

Dr. Walter has been an invited member and reviewer for a number of committees in the area of fault tolerance and computer systems. He was a member of the Fault Tolerant Task Group for the Navy's Next Generation Computing Resources (NGCR) program, under contract to evaluate proposed standards for dependability attributes and their applicability to the design of mission critical computing applications. He has served as a reviewer for a number of technical program committees. In particular he was a reviewer for NASA research on Formal Methods for real-time systems, National Science Foundation, and the Swedish Industrial Board (NUTEK).

Patrick Lincoln received a BS degree from MIT and a PhD from Stanford, both in Computer Science. He is the director of the Computer Science Laboratory at SRI International, where he has worked since 1989. He has also performed research at MCC in Austin, Texas, and Los Alamos National Laboratory. His research interests include formal methods, fault tolerance, rewriting, linear logic, logics of programming, and computer security. See http://www.csl.sri.com/~lincoln for more information.

Neeraj Suri received his MS and PhD from the University of Massachusetts at Amherst. He is a faculty member in the Department of Computer and Information Sciences at NJIT, and also the co-director of the Dependable Real-time Systems Lab at NJIT.

His research interests focus on the design and analysis of algorithms for the provision of dependable, distributed real-time system services, and on the processes for ensuring the correctness and completeness properties of the developed algorithms. His research looks at the use of formal techniques for verification and validation (V&V) of dependability, including a novel extension on the use of formal methods in directing fault-injection based validation. Recently, he been developing formal specification and verification approaches dealing with discrete time, and especially continuous time domains, for V&V of real-time services.

The website http://www.cs.njit.edu/~suri provides more information on his interests and activities.

Dr. Suri is a senior member of the IEEE.