A Formally Verified Algorithm for Interactive Consistency Under a Hybrid Fault Model[*]

Patrick Lincoln and John Rushby
Computer Science Laboratory
SRI International
Menlo Park CA 94025 USA

Abstract

Thambidurai and Park [19] introduced a "hybrid" fault model that includes the arbitrary (or Byzantine) fault mode, together with two simpler and common fault modes. They also introduced a modified version of the "Oral Messages" algorithm for Interactive Consistency (Byzantine Agreement) that provides the best of both worlds by tolerating more simple faults than the original, while retaining its ability to withstand arbitrary faults. Unfortunately, despite a published proof of correctness, their algorithm is flawed and fails in quite simple circumstances.

We detected this flaw while undertaking a formal verification of the algorithm using our PVS mechanical verification system [13]. Repairing this algorithm is not easy. We developed an incorrect version ourselves, and even "proved" it correct using ordinary, informal mathematics.

The discipline of mechanically-checked formal verification eventually enabled us to develop a correct algorithm for Interactive Consistency under the hybrid fault model. In the paper, we present this algorithm, discuss its subtle points, and describe its formal specification and verification. We argue that formal verification systems such as PVS are now sufficiently effective that their application to fault-tolerance algorithms should be considered routine.

Keywords:
interactive consistency, Byzantine agreement, hybrid fault models, formal verification.

[*]This work was supported by the National Aeronautics and Space Administration, Langley Research Center, under contract NAS1-18969.

1 Introduction

Fault-tolerant systems are designed and evaluated against explicit assumptions regarding the kinds and numbers of faults they are to tolerate. "Fault models," which enumerate the assumed behaviors of faulty components, range from those that identify many highly specific modes of failure, to those that comprise just a few broad classes. The advantage of a very detailed fault model is that the mechanisms of fault tolerance can be finely tuned to deliver maximum resilience from a given level of redundancy; the corresponding disadvantages are that an overlooked fault mode may cause unexpected failure in operation, and the need to counter many fault modes can lead to a complex design—which may itself be a source of faults.

In contrast to designs that consider many fault modes are those that make no assumptions whatsoever about the behavior of faulty components. The advantage of such "Byzantine" fault-tolerant designs is that they cannot be defeated by unexpected failure modes; their disadvantage is that all faults are treated as "worst case," so that large levels of redundancy tolerate relatively few faults. For example, a conventional Byzantine fault-tolerant architecture requires $3m+1$ channels to tolerate m simultaneous faults of any kind within some of its functions [1, 14]. Thus, a 4-plex is needed in order to withstand a single fault,[1] and 5- and 6-plexes provide no additional benefit (in fact the additional channels will increase the fault arrival rate and thereby lower overall reliability).[2] This seems counterintuitive, since it is clear that suitably organized 5- and 6-plexes can withstand more faults, of some kinds, than a 4-plex.

[1]Single fault tolerance can also be provided by architectures such as Draper Laboratory's FTP, which uses only three full processors plus three simpler "interstages" [4].

[2]If the system can be reconfigured following a fault, then a fifth and sixth channel can increase reliability if used as standby spares—but they serve no purpose as live channels.

These observations motivate the study of fault-tolerant architectures and algorithms with respect to "hybrid" fault models that include the Byzantine, or "arbitrary," fault mode, together with a limited number of additional fault modes. Inclusion of the arbitrary fault mode (i.e., faults whose behaviors are entirely unconstrained) eliminates the fear that some unforeseen mode may defeat the fault-tolerance mechanisms provided, while inclusion of other fault modes allows greater resilience to be achieved for faults of these kinds than with a classical Byzantine fault-tolerant architecture.

Our interest is architectures for digital flight-control systems, where fault-masking behavior is required to ultra-high levels of reliability. This means that not only must stochastic modeling show that adequate numbers and kinds of faults are masked to satisfy the mission requirements, but that convincing analytical evidence must attest to the soundness of the overall fault-tolerant architecture and to the correctness of the design and implementation of its mechanisms of fault tolerance.[3] Such a rational design for a "reliable computing platform" suitable for ultra-critical applications was established in the late 1970s and early 1980s by the SIFT architecture [21]: the system workload is executed by several independent channels operating in approximate synchrony, and results are subjected to majority voting. If all channels execute identical workloads on identical data, then majority voting is sufficient to mask arbitrary channel failures. However, majority voting is not sufficient to mask arbitrary failures in the distribution of single-source data (such as sensor samples) [14], nor in clock synchronization [6].

In this paper, we focus on algorithms for reliably distributing single-source data to multiple channels in the presence of faults. This problem, known as "Interactive Consistency" (although sometimes called "source congruence"), was first posed and solved for the case where faulty channels can exhibit arbitrary behavior by Pease, Shostak, and Lamport [14] in 1980.[4] Interactive Consistency is a symmetric problem: it is assumed that each channel has a "private value" (e.g., a set of sensor samples) and the goal is to ensure that every nonfaulty channel achieves an accurate record of the private value of every other nonfaulty channel. In 1982, Lamport, Shostak, and Pease [7] presented an asymmetric version of Interactive Consistency, which they called the "Byzantine Generals Problem"; here, the goal is to communicate a single value from a designated channel called the "Commanding General" to all the other channels, which are known as "Lieutenant Generals." The problem of real practical interest is Interactive Consistency, but the metaphor of the Byzantine Generals has proved so memorable that this formulation is better known; it can also be easier to describe algorithms informally using the Byzantine Generals formulation, although the balance of advantage can be reversed in truly formal presentations [15]. An algorithm for the Byzantine Generals problem can be converted to one for Interactive Consistency by simply iterating it over all channels (each channel in turn taking the role of the Commander), so there is no disadvantage to considering the Byzantine Generals formulation. See [15] for more extended discussion of this topic.

Lamport, Pease, and Shostak presented algorithms for solving the Byzantine Generals problem. The principal difficulty to be overcome in such algorithms is possibly asymmetric behavior on the part of faulty channels: such a channel may provide one value to a second channel, but a different value to a third, thereby making it difficult for the recipients to agree on a common value. Byzantine Generals algorithms overcome the possibility of faulty channels exhibiting asymmetric behavior by using several rounds of message exchange during which channel p tells channel q what value it received from channel r and so on. The precise form of the algorithm depends on assumptions about what a faulty channel may do when relaying such a message; under the "Oral Messages" assumption, there is no guarantee that a faulty channel will relay messages correctly. This corresponds to totally arbitrary behavior by faulty channels: not only can a faulty channel provide inconsistent data initially, but it can also relay data inconsistently.[5]

Using $m + 1$ rounds of message exchanges, the Oral Messages algorithm of Lamport, Shostak, and Pease [7], which we denote $\mathrm{OM}(m)$, can withstand up to m arbitrary faults, provided n, the number of channels, satisfies $n > 3m$. The bound $n > 3m$ is optimal: Pease, Shostak, and Lamport proved that no algorithm based on the Oral Messages assumptions can withstand more arbitrary faults than this [14]. However, as we have already noted, $\mathrm{OM}(m)$ is not optimal when other than arbitrary faults are considered: other

[3]There are examples where unanticipated behaviors of the mechanisms for fault tolerance became the *primary* source of system failure [10].

[4]Davies and Wakerley had anticipated some of the issues a few years earlier [2].

[5]Under the "signed messages" assumption (which can be satisfied using digital signatures), an altered message can be detected by the recipient.

algorithms can withstand greater numbers of simpler faults for a given number of channels than OM(m).

We are not the first to make these observations. Thambidurai and Park [19] and Meyer and Pradhan [11, 12] have considered Interactive Consistency algorithms that resist multiple fault classes. Thambidurai and Park's "Unified" model divides faults into three classes: *nonmalicious* (or *benign*), *symmetric malicious*, and *asymmetric malicious*. A nonmalicious fault is one that produces detectably missing values (e.g., timing, omission, or crash faults), or that produces a "self-incriminating" value that all nonfaulty recipients can detect as bad (e.g., it fails checksum or format tests). A malicious fault is one that yields a value that is not detectably bad (i.e., it is a *wrong*, rather than a missing or manifestly corrupted, value). A symmetric malicious fault delivers the same wrong value to every nonfaulty receiver; an asymmetric malicious fault delivers (possibly) different wrong values (or missing or detectably bad values) to different nonfaulty receivers. The classical arbitrary or Byzantine fault is an asymmetric malicious fault in this taxonomy.[6]

Thambidurai and Park present a variant on the classical Oral Messages algorithm that retains the effectiveness of that algorithm with respect to arbitrary faults, but that is also capable of withstanding more faults of the other kinds considered.[7] In a later paper, Thambidurai, Park, and Trivedi [20] present reliability analyses that show this increased fault-tolerance indeed provides superior reliability under plausible assumptions. McElvany-Hugue has also studied the reliability of related algorithms under this fault model, reaching similar conclusions [5].

Unfortunately, Thambidurai and Park's algorithm (which they call Algorithm Z) has a serious flaw and fails in quite simple circumstances. In this paper, we describe the flaw, and explain how straightforward attempts to repair it also fail. We then present a correct algorithm for the problem of Interactive Consistency under a hybrid fault model and present a proof of its correctness. Thambidurai and Park presented a proof of correctness for their flawed algorithm, and we have also developed some rather convincing "proofs" ourselves for other incorrect algorithms for this problem. Accordingly, we have developed a mechanically checked formal verification for our algorithm using the PVS verification system [13]. We outline this formal verification and show that it is not particularly difficult. Because informal proofs seem unreliable in this domain, and the consequences of failure could be catastrophic, we argue that formal verification should become routine.

[6]Thambidurai and Park state that "it is possible for a fault to be symmetric or asymmetric and still be nonmalicious ... our unified model is capable of treating both cases uniformly. A nonmalicious asymmetric fault is no worse than a symmetric fault" [19, page 96]. This statement seems to indicate that an asymmetric nonmalicious fault need not be detected by every receiver; this is not so, however, as scrutiny of their proof reveals. The intended interpretation seems to be that an asymmetric nonmalicious fault may result in different, but still manifestly erroneous, values being received by different channels, whereas all values must be the same in the symmetric case.

[7]Meyer and Pradhan [12] consider a fault model that, in Thambidurai and Park's taxonomy, comprises only asymmetric malicious and benign faults. Their algorithm is derived from the algorithm of [3] and, like the parent algorithm, is not particularly suitable for the cases of practical interest (i.e., $m = 1$, or possibly $m = 2$, n less than 10).

2 Requirements, Assumptions, and the Algorithms OM and Z

Although the problem of real practical interest is Interactive Consistency, all the algorithms we consider are presented here in their Byzantine Generals formulation, since this appears simpler in informal presentations. The relationship between Interactive Consistency and the Byzantine Generals Problem is examined in [15].

2.1 Requirements

In the Byzantine Generals formulation of the problem, there are n participants, which we call "processors." A distinguished processor, which we call the *transmitter*, possesses a value to be communicated to all the other processors, which we call the *receivers*.[8] There are n processors in total, of which some (possibly including the transmitter) may be faulty. The transmitter's value is denoted v and 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 conditions:

BG1: If receivers p and q are nonfaulty, then they agree on the value ascribed to the transmitter—that is, for all nonfaulty p and q, $\nu_p = \nu_q$.

BG2: If the transmitter is nonfaulty, then every nonfaulty receiver computes the correct value—that is, for all nonfaulty p, $\nu_p = v$.

[8]Lamport, Shostak, and Pease [7] speak of a "Commanding General" and of "Lieutenant Generals" where we say transmitter and receivers.

Conditions BG1 and BG2 are sometimes known as "Agreement" and "Validity," respectively.

2.2 Assumptions

The principal difficulty that must be overcome by a Byzantine Generals algorithm is that the transmitter may send different values to different receivers, thereby complicating satisfaction of condition BG1. To overcome this, 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. More precisely, the *Oral Messages* assumptions are the following.

A1: Every message that is sent between nonfaulty processors is correctly delivered.

A2: The receiver of a message knows who sent it.

A3: The absence of a message can be detected.

In the classical Byzantine Generals problem, there are no constraints at all on the behavior of a faulty processor.

2.3 Algorithm OM

Lamport, Shostak, and Pease's Algorithm OM solves the Byzantine Generals problem under the Oral Messages assumption. The algorithm is parameterized by m, the number of rounds of message exchanges performed. OM(m) can withstand up to m faults, provided $n > 3m$, where n is the total number of processors. The algorithm is described recursively; the base case is OM(0).

OM(0)

1. The transmitter sends its value to every receiver.
2. Each receiver uses the value obtained from the transmitter, or some arbitrary, but fixed, value if nothing is received.

Now we can describe the general case.

OM(m), m > 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 else be some arbitrary, but fixed, value if it obtains no value. Each receiver p acts as the transmitter in Algorithm OM($m-1$) to communicate its value v_p to each of the $n-2$ other receivers.
3. For each p, and each $q \neq p$, let v_q be the value receiver p obtained from receiver q in step (2) (using Algorithm OM($m-1$)), or else some arbitrary, but fixed, value if nothing was received. Each receiver p calculates the majority value among all values v_q it receives, and uses that as the transmitter's value (or some arbitrary, but fixed, value if no absolute majority exists).

The correctness of this algorithm (that it achieves BG1 and BG2 under certain assumptions) was proven in [7, page 390] and mechanically checked in [1,15].

2.4 Algorithm Z

Thambidurai and Park's Algorithm Z is a modification of OM intended to operate under their hybrid fault model described earlier. The difference between OM and Z is that the latter has a distinguished "error" value, E. Any processor that receives a missing or manifestly bad value replaces that value by E and uses E as the value that it passes on in the recursive instances of the algorithm. The majority voting that is required in OM, is replaced in Z by a majority vote with all E values eliminated. Thambidurai and Park claim that an m-round implementation of Algorithm Z can withstand $a+s+b$ simultaneous faults, where a is the number of asymmetric malicious faults, s the number of symmetric malicious faults, and b the number of nonmalicious faults, provided $a \leq m$, and n, the number of processors, satisfies $n > 2a + 2s + b + m$. In the case of no symmetric malicious or nonmalicious faults (i.e., Byzantine faults only), we have $m = a$ and $s = b = 0$, so that $n > 3m$ and the algorithm provides the same performance as the classical Oral Messages algorithm.

We and our colleagues at SRI have undertaken mechanically checked formal verifications for a number of fault-tolerant algorithms, including OM [15], and have identified deficiencies in some of the previously published analyses (though not in the algorithms—see, for example [17,18]). Any changes to the established algorithms for Interactive Consistency must be subjected to intense scrutiny, for errors in these algorithms are single points of failure in any system that employs them. Changes that widen the classification of faults considered are likely to increase the case analysis, and hence the complexity and potential fallibility of arguments for the correctness of modified algorithms. We therefore considered Thambidurai and Park's Algorithm Z an interesting candidate for formal verification.

We began our attempt to formally verify Algorithm Z by studying the proof of its correctness provided by Thambidurai and Park [19, pages 96 and 97]. This proof follows the outline of the standard proof for OM [7, page 390] quite closely. However, we soon found that Thambidurai and Park's proof is simultaneously more complicated than necessary and flawed in several details. The most serious fault is that their Lemma 1 (all nonfaulty receivers get the correct value of a nonfaulty transmitter) fails to consider the case where the value sent by the transmitter is E. This can arise in recursive instances of the algorithm when nonfaulty receivers are passing on the value received from a faulty source. Further thought soon reveals that not only is the proof flawed, but the algorithm is incorrect: even systems with large numbers of processors may fail with only two faulty components.

The simplest counterexample comprises five processors in which the transmitter has a nonmalicious fault, one of the receivers has an asymmetric malicious fault, and the algorithm is Z with one round (i.e., $n = 5, a = 1, s = 0, b = 1, m = 1$). All the nonfaulty receivers note E as the value received from the transmitter, and relay the value E to all the other receivers. The faulty receiver sends a different (non-E) value to each of the nonfaulty receivers. Each nonfaulty receiver then has three E values, and one non-E value; because E values are discarded in the majority vote, each nonfaulty receiver selects the value received from the faulty receiver as the value sent by the transmitter. Since these values are all different, the algorithm has failed to achieve agreement among the nonfaulty receivers. Observe that this scenario is independent of the number of receivers (provided there are more than three of them—two that should agree and one that is faulty), so the problem is not due to an inadequate level of redundancy.

3 The Algorithm OMH

In this section we introduce our new algorithm *OMH* for interactive consistency under a hybrid fault model. Before describing the algorithm, we present the fault model.

3.1 Hybrid Fault Model

Our fault model is that of Thambidurai and Park, but with the cases renamed—we find the anthropomorphism in terms such as "malicious faults" unhelpful.

The fault modes we distinguish for processors are *arbitrary-faulty*, *symmetric-faulty*, and *manifest-faulty* (also called *crash-faulty*). (These correspond to Thambidurai and Park's asymmetric malicious, symmetric malicious, and nonmalicious faults, respectively.) Of course, we also need a class of *good* (also called *nonfaulty*) processors. We specify these fault modes semiformally as follows (the formal characterizations are presented in the following section).

Memo: Is it?

When a transmitter sends its value v to the receivers, the value obtained by a nonfaulty receiver p is:

- v, if the transmitter is nonfaulty
- E, if the transmitter is manifest-faulty[9]
- Unknown, if the transmitter is symmetric-faulty, but all receivers obtain the *same* value,
- Completely unconstrained, if the transmitter is arbitrary-faulty.

Note that it is not necessary to define the value received by a faulty receiver, because such receivers may send values completely unrelated to their inputs. Also note that manifest faults must be symmetric. If a processor were to "crash" during this protocol (or if some of its outgoing links are broken, or if it is early or late transmitting on some links), it would have to be counted as arbitrary-faulty, since different good receivers may obtain different values—even though the values sent are either correct or identifiably bad. It is possible to treat such cases as a new class of faults, which, depending on practical considerations, may be an interesting area for future research.

[9]Some preprocessing of timeouts, parity and "reasonableness" checks, etc. may be necessary to identify manifestly faulty values. The intended interpretation is that the receiver detects the incoming value as missing or bad, and then replaces it by the distinguished value E.

3.2 The Algorithm

It seems that the flaw in Algorithm Z stems from the fact that it does not distinguish between values received from manifest-faulty processors and the *report* of such values received from nonfaulty processors; the single value E is used for both cases. Thus, a plausible repair for Algorithm Z introduces an additional distinguished value RE (for Reported Error); when a manifestly faulty value is received, the receiver notes it as E, but passes it on as RE; if an RE is received, it is noted and passed on as such. Only E values are discarded when the majority vote is taken. In the counterexample to Algorithm Z given above, the nonfaulty receivers in this modified algorithm will each interpret the value received from the transmitter as E, and pass it on to the other receivers as RE. In their majority votes, each nonfaulty receiver has a single E (from the transmitter), which it discards, two REs (from the other nonfaulty receivers), and an arbitrary value (from the faulty receiver). All will therefore select RE as the value ascribed to the transmitter.

Unfortunately this modified algorithm has two defects. First, a receiver that obtains a manifest-faulty value from the transmitter notes it as E, but passes it on as RE. Thus, this receiver will omit the value from its majority vote, but the others will include it (as RE). This asymmetry can be exploited by an arbitrary-faulty transmitter to force the receivers into disagreement (consider an arbitrary-faulty transmitter and three nonfaulty receivers, where the transmitter sends the values E, RE, and a normal value).

It therefore seems that receivers must distinguish between an E received from the transmitter (which must be treated locally as RE and passed on as such), and one received from another receiver (which can be discarded in the majority vote). This repair fixes one problem, but leaves the other: the value ascribed to a manifest faulty transmitter is not E, but RE. This might seem a small inconvenience, but it causes the algorithm to fail when m, the number of rounds, is greater than 1 (consider the case $n = 6$, $m = 2$ when there is a nonfaulty transmitter and three manifest-faulty receivers).

A repair to this difficulty might be to return the value E whenever the majority vote yields the value RE. This modification has the problem that receivers cannot distinguish a manifest-faulty receiver from a nonfaulty one reporting that another is manifest-faulty (consider the case $n = 4$, $m = 1$, all the processors are nonfaulty, and the transmitter is trying to send RE—as can arise in recursive cases when $m > 1$).

Like Thambidurai and Park did for Algorithm Z, we produced rather convincing, but nonetheless flawed, informal "proofs of correctness" for these erroneous repairs to Algorithm Z. Eventually, the discipline of formal verification (where one must deal with the implacable skepticism of a mechanical proof checker and is eventually forced to confront overlooked cases and unstated assumptions) enabled us to develop a genuinely correct algorithm for this problem.

Our new algorithm, OMH (for "Oral Messages, Hybrid"), is somewhat related to the last of the modifications to Algorithm Z indicated above, but recognizes that a single "reported error" value is insufficient. OMH employs two functions R and UnR that act as a "wrapper" and an "unwrapper" for error values.

The basic idea of OMH is that at each round, the processors do not forward the actual value they received. Instead, each processor sends a value corresponding to the statement "I'm reporting *value*." One can imagine that after several rounds, messages corresponding to "I'm reporting that he's reporting that she's reporting an Error value" arise. This wrapper is only required for error values, but for simplicity we assume that the functions R and UnR are applied to *all* values. Alternatives to this are explored in Section 4. This leaves the following intuitive picture of the algorithm.

Proceed as in the usual OM Byzantine agreement algorithm presented above, with the following exceptions. Add a distinguished error value E, and two functions on values R and UnR. When a manifestly bad value is received, temporarily record it as the special value E. When passing along a value received from the transmitter or incorporating it into the local majority vote, apply R, standing for "I report..." to the value. Discard all E values (received from other receivers) before voting, but treat all other error values ($R(E)$, $R(R(E))$, etc.) as normal, potentially valid values during voting. After voting, apply UnR (strip off one R) before returning the value.

The key idea here is that in Z and related algorithms there is a confusion about which processors have manifest faults: if there is only one error value, E, how can a processor distinguish between a manifest-faulty receiver and a good receiver reporting a bad value (or the lack of a value) from a manifest-faulty transmitter? The counterexample to Algorithm Z given above exploits this confusion, but it is handled correctly by OMH, because the nonfaulty receivers in OMH(1) each receive a single E from the transmitter, which they pass on to the other receivers and themselves as $R(E)$. The values thus voted on include three $R(E)$s and an arbitrary value (from the arbitrary-faulty receiver). All nonfaulty receivers therefore select $R(E)$ as the majority value. After stripping one R from this value, the result correctly identifies the transmitter as manifest-faulty. In short, OMH incorporates the diagnosis of manifest faults into the agreement algorithm.

The Hybrid Oral Messages Algorithm OMH(m) is defined more formally below. A completely formal specification is given in Appendix A.

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 manifestly erroneous value is received.

OMH(m), m > 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 manifestly 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($m-1$)), or else E if no such value, or a manifestly bad value, was received. Each receiver p calculates the majority value among all non-error 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.

3.3 Semiformal Notation and Proofs

We make explicit a few unsurprising technical assumptions:

- All processors are either nonfaulty, arbitrary-faulty, symmetric-faulty, or manifest-faulty. (Any fault not otherwise classified is considered arbitrary.)
- Processors do not change fault status during the procedure; for example, if a nonfaulty processor were to become manifest-faulty during this procedure, we would say that processor is arbitrary-faulty because it has effectively sent different values to other processors.
- For all values v, $R(v) \neq E$. (Wrapped values are never mistaken for errors.)
- For all values v, $UnR(R(v)) = v$. (Unwrapping a wrapped value results in the original value.)

Algorithm OMH must satisfy the Byzantine Generals conditions naturally extended to the fault model described above.

When the transmitter is symmetric-faulty, it is convenient to call the unique value received by all nonfaulty receivers the value *actually sent* by the transmitter.

BGH1: If processors p and q are nonfaulty, then they agree on the value ascribed to the transmitter; that is, $\nu_p = \nu_q$.

BGH2: If processor p is nonfaulty, the value ascribed to the transmitter by p is

- The correct value v, if the transmitter is nonfaulty,
- The value actually sent, if the transmitter is symmetric-faulty,
- The value E, if the transmitter is manifest-faulty.

The argument for the correctness of OMH is an adaptation of that for the Byzantine Generals formulation of OM [7, page 390]. We define

- n, the number of processors,
- a, the maximum number of arbitrary-faulty processors the algorithm is to tolerate,
- s, the maximum number of symmetric-faulty processors the algorithm is to tolerate,

- c, the maximum number of manifest-faulty processors the algorithm is to tolerate,[10]

- m, the number of rounds the algorithm is to perform.

Lemma 1 *For any a, s, c and m, Algorithm OMH(m) satisfies BGH2 if there are more than $2(a+s)+c+m$ processors.*

Proof: The proof is by induction on m. BGH2 specifies only what must happen if the transmitter is not arbitrary-faulty. In the base case $m = 0$, a nonfaulty receiver obtains the transmitter's value if the transmitter is nonfaulty. If the transmitter is symmetric-faulty the value obtained is the value actually sent. If the transmitter is manifest-faulty the receiver obtains the value E. So the trivial algorithm OMH(0) works as advertised and the lemma is true for $m = 0$. We now assume the lemma is true for $m-1$ ($m > 0$), and prove it for m.

In step (1) of the algorithm, the transmitter effectively sends some value ν to all $n-1$ receivers. If the transmitter is nonfaulty, ν will be v, the correct value; if it is symmetric-faulty, ν is the value actually sent; if it is manifest-faulty, ν is E. In any case, we want all the nonfaulty receivers to decide on ν.

Memo: Hybrid-majority doesn't seem to be defined before it's used.

In step (2), each receiver applies OMH($m-1$) with $n-1$ participants. Those receivers that are nonfaulty will apply the algorithm to the value $R(\nu)$. Since by hypothesis $n > 2(a+s)+c+m$, we have $n-1 > 2(a+s)+c+(m-1)$, so we can apply the induction hypothesis to conclude that the nonfaulty receiver p gets $v_q = R(\nu)$ for each nonfaulty receiver q. Let c' denote the number of manifest-faulty processors among the receivers. At most $(a+s+c')$ of the $n-1$ receivers are faulty, so each nonfaulty receiver p obtains a minimum of $n-1-(a+s+c')$ values equal to $R(\nu)$. Since there are c' manifest-faulty processors among the receivers, a nonfaulty receiver p also obtains a minimum of c' values equal to E and, therefore, at most $n-1-c'$ values different from E. The value $R(\nu)$ will therefore win the *hybrid-majority* vote performed by each nonfaulty processor p, provided

$$2(n-1-(a+s+c')) > n-1-c',$$

that is, provided

$$n > 2(a+s)+c'+1.$$

Now, $c' \le c$, and $1 \le m$, so this condition is ensured by the constraint

$$n > 2(a+s)+c+m.$$

Finally, *UnR* is applied to the result $R(\nu)$, which results in final value ν. □

Theorem 1 *For any m, Algorithm OMH(m) satisfies conditions BGH1 and BGH2 if there are more than $2(a+s)+c+m$ processors and $m \ge a$.*

Proof: The proof is by induction on m. In the base case $m = 0$ there can be no arbitrary-faulty processors, since $m \ge a$. If there are no arbitrary-faulty processors then the previous lemma ensures that OMH(0) satisfies BGH1 and BGH2. We therefore assume that the theorem is true for OMH($m-1$) and prove it for OMH(m), $m > 0$.

We next consider the case in which the transmitter is not arbitrary-faulty. Then BGH2 is ensured by Lemma 1, and BGH1 follows from BGH2.

Now consider the case where the transmitter is arbitrary-faulty. There are at most a arbitrary-faulty processors, and the transmitter is one of them, so at most $a-1$ of the receivers are arbitrary-faulty. Since there are more than $2(a+v)+c+m$ processors, there are more than $2(a+s)+c+m-1$ receivers, and

$$2(a+s)+c+m-1 > 2([a-1]+s)+c+[m-1].$$

We may therefore apply the induction hypothesis to conclude that OMH($m-1$) satisfies conditions BGH1 and BGH2. Hence, for each q, any two nonfaulty receivers get the same value for v_q in step (3). (This follows from BGH2 if one of the two receivers is processor q, and from BGH1 otherwise). Hence, any two nonfaulty receivers get the same vector of values $v_1, \ldots, v_{n-1}$, and therefore obtain the same value *hybrid-majority*$(v_1, \ldots, v_{n-1})$ in step (3) (since this value is functionally determined), thereby proving BGH1. □

[10]We cannot use m for the number of manifest-faulty processors, because the parameter m is traditionally used for the number of rounds (although Thambidurai and Park use r). The symbol c can be considered a mnemonic for "crashed," which is one of the failures that can generate manifest-faulty behavior.

3.4 Benefits

Recall that OM achieves agreement and validity if there are more than three times as many good processors as arbitrary-faulty processors ($n > 3a$). From the bounds given in Theorem 1, $n > 2(a+s)+c+m$ and $m \geq a$, it may be seen that OMH achieves the same resilience to arbitrary faults if there are no symmetric-faulty or manifest-faulty processors.

Memo: Also, from Theorem 2, if $a = s = 0$, then all that is required is that $n > c$.

Number of Faults		
Arbitrary (a)	Symmetric (s)	Manifest (c)
1	1	0
1	0	2
0	2	0
0	1	2
0	0	5

Table 1: Fault-Masking Capabilities of OMH(1) with $n = 6$

Memo: Need to explain the final 5.

Table 1 indicates the different numbers of simultaneous faults that a 6-plex can withstand using OMH(1); for comparison, observe that OM(1) can withstand only a single (arbitrary) fault in this configuration. Thambidurai, Park and Trivedi [20] present reliability analyses that show this increased fault-tolerance indeed provides superior reliability under plausible assumptions[11].

Memo: Following is moved in report

Although a major improvement on OM, the number of faults that can be tolerated by OMH according to the analysis given above is clearly not optimal in some of the extreme circumstances. In some cases, the algorithm is suboptimal, in others, the general analysis above is too conservative. As an example of the latter, consider the case where only manifest faults are present. In this case, the general analysis shows that the number of faults that can be tolerated is $n-m-1$: in other words, the greater the number of rounds, the *less* manifest faults can be tolerated. In fact, alternate analysis shows that OMH(m) tolerates the maximum possible number (i.e., $n-1$) of manifest-faulty processors when there are no arbitrary nor value faults.

Theorem 2 *If arbitrary and value-faults are not present, Algorithm OMH(m) satisfies conditions BGH1 and BGH2 if there are more than c processors.*

This theorem has been formalized and mechanically verified, but we do not give the details of the proof here.

When only value-faults are present, however, it is the algorithm, rather than its general analysis, that is less than optimal. Here, the additional rounds of message exchanges are actively counter-productive in the cases $m > 0$ (compare $n = 4$, $v = 2$ for the cases $m = 0$ and $m = 1$). Additional rounds of messages are the price paid for overcoming arbitrary faults, and these seem to reduce the ability to deal with value faults. An interesting topic for future research is to investigate whether this trade-off can be mitigated.

4 Practical Considerations

The OMH algorithm has been presented in an abstract form to facilitate formal specification, verification, and presentation. Additional practical considerations have been incorporated in alternate formal specifications, and are briefly discussed here. We have not formally verified these modified versions.

Possible low-level implementations of R and UnR are increment and decrement operations, assuming all data values are integers and E is assigned one less than the smallest data value. Intermediate error values such as $R(R(E))$ may overlap with other actual values (such as 2), but the final result remains correct because UnR (decrement) is always applied to the output of the majority vote.

Note that the requirements on R and UnR (for all v, $R(v) \neq E$, and $UnR(R(v)) = v$), impose an infinite family of values, E, $R(E)$, $R(R(E))$, ... However, for an m round OMH, just $m+1$ error values (E

[11]Although algorithm Z is somewhat flawed, the analysis in [20] can be correctly applied to OMH

up to $R^m(E)$) suffice with the following modifications to the algorithm. Add a test to the OMH algorithm comparing the number of applications of R with the depth of recursion in OMH. (This is not the same as $R^{x \bmod m}(E)$.) Any value with more Rs than elapsed rounds may be replaced by E. In the common case of one round OMH, two error values, E and $R(E)$, suffice. The above low-level implementation may be used with this modified algorithm if the largest $m+1$ data values are reserved.

Using these techniques, one may reduce the overhead of using OMH-like algorithms (as compared to OM) to a small constant number of extra data values, and a slightly more complex algorithm. The message complexity of OMH, and all OM-like algorithms can be quite large (exponential) in the limiting case, but this is of less concern for implementations using small numbers of rounds, and there may be modifications to OMH which reduce this message complexity a great deal.

5 Formal Specification and Verification of OMH

The OMH(n) algorithm and properties have been formally specified and verified using the PVS verification system [13]. The formal specification of OMH was derived from one for the classical OM algorithm [15] and was iteratively developed as failed attempts at formal verification exposed the errors described earlier. In more than one case the authors developed convincing informal arguments and attempted to verify the claims using PVS. The tirelessly skeptical theorem prover would not accept these flawed arguments, and eventually led us to discover counterexamples in these supposed algorithms. Finally, we were able to develop the new algorithm presented above, and prove that correct. The specification and verification are described in detail elsewhere [8, 9].

The greatest benefit of the formal specification and verification in this case has been the refinement of our own understanding. It is very easy for humans to be convinced to the correctness of flawed algorithms in complex and unnatural domains. It has been argued that a large part of the added value of formal methods lies in specification. Formal specifications serve as a prod to clarify ones thinking as well as provide a means of communication less subject to error and misinterpretation than traditional documentation. However, the additional step of verification has proved critical in this case. Our formal specification of the flawed algorithms only served to firm our erroneous convictions about them. Only through failed attempts at formal verification were some of the errors detected.

The further step of validating the specification is also suggested; the assumptions of a formal specification must match the intended application. Peer review is perhaps the best method of ensuring that a set of assumptions are true about the intended application, but another tool we use is putative theorem proving. One formulates theorems that "should" be true if the specification matched the application, and verifies them. In the application here, for example, one might hope that a message sent from a nonfaulty channel to itself is always received correctly, and that the hybrid majority of a collection of values not containing E is the same as the simple majority of that collection. We produced one specification of a flawed algorithm for which we were able to prove the validity property. However, one of the axioms about hybrid majority was stated incorrectly. Only through attempting to prove putative theorems and refine the specification did the inconsistency of the axioms become apparent.

Full PVS specifications and PVS proofs are available from the authors or by anonymous FTP from FTP.CSL.SRI.COM.

FOR THE CONCLUSION:

We are not interested in pushing the envelope of size and complexity of verified systems. Our interest is in providing added value to the system validation process. In this vein we have chosen particularly critical and troublesome aspects of system operation.

...Although the cost of formal verification is still high,

6 Conclusions

Thambidurai and Park's hybrid fault model extends the design and analysis of Byzantine fault-tolerant algorithms in an important and useful way. Hybrid fault-tolerant algorithms can tolerate greater numbers of "simple" faults than classical Byzantine fault-tolerant algorithms, without sacrificing the ability to withstand Byzantine, or arbitrary, faults. Unfortunately, their Algorithm Z for achieving Interactive Consistency under a hybrid fault model is incorrect. In the preceding sections, we have described the problem with Algorithm Z and presented OMH, a correct algorithm for this problem.

Memo: We applied our formal verification tools to this domain, discovering errors in published proofs and in a proposed algorithm for Byzantine Agreement under this fault model.

A crucial tool in our detection of the flaw in Thambidurai and Park's algorithm, and also in detecting flaws in our own early attempts to repair this algorithm, was our use of mechanically-checked formal verification. The discipline of formal specification and verification was also instrumental in helping us to develop the correct algorithm presented here. The rigor of a mechanically-checked proof enhances our conviction that this algorithm is, indeed, correct, and also helped us develop the informal, but detailed, proof given here in the style of a traditional mathematical presentation.

It is worth repeating that no formal verification proves any program "correct". At most, a program is shown to satisfy a specification, and shown to exhibit certain properties under a whole host of assumptions about the context in which the program is run. The true benefit of formal specification and verification is *not* in getting a theorem prover to say `proved`, but rather in refining one's understanding through dialogue with a tirelessly skeptical theorem prover.

The effort required to perform this formal verification was not particularly large and did not seem to us to demand special skill. We attribute some of this ease in performing formal verification of a relatively tricky algorithm to the effectiveness of the tools employed [13]. These tools (and others that may be of similar effectiveness) are freely available. In light of the flaws we discovered in Thambidurai and Park's algorithm, and had previously found in the proofs for other fault-tolerant algorithms [17,18], we suggest that formal verification should become a routine part of the social process of development and analysis of fault-tolerant algorithms intended for practical application in safety-critical systems.

In future work, we hope to explore possible extensions to the OMH algorithm and its analysis to include communication faults, and to see whether larger numbers of symmetric faults can be tolerated. We also intend to study whether lower message complexity can be achieved in cases of practical interest, and to examine alternative architectures employing fewer processors (we have already formally specified and verified a variant of OMH(1) for the asymmetric Draper FTP architecture [4]). We also plan to formally verify a modified version of the Interactive-Convergence Algorithm for clock synchronization using a hybrid fault model that includes communication faults (we have already formally verified the standard algorithm [16], and have an informal analysis of the modified version).

References

[1] William R. Bevier and William D. Young. Machine-checked proofs of the design and implementation of a fault-tolerant circuit. NASA contractor report 182099, NASA Langley Research Center, Hampton, VA, November 1990. (Work performed by Computational Logic Incorporated).

[2] Daniel Davies and John F. Wakerly. Synchronization and matching in redundant systems. *IEEE Transactions on Computers*, C-27(6):531–539, June 1978.

[3] Danny Dolev, Michael J. Fischer, Rob Fowler, Nancy A. Lynch, and H. Raymond Strong. An efficient algorithm for Byzantine agreement without authentication. *Information and Control*, 52:257–274, 1982.

[4] Albert L. Hopkins, Jr., Jaynarayan H. Lala, and T. Basil Smith III. The evolution of fault tolerant computing at the Charles Stark Draper Laboratory, 1955–85. In A. Avižienis, H. Kopetz, and J. C. Laprie, editors, *The Evolution of Fault-Tolerant Computing*, volume 1 of *Dependable Computing and Fault-Tolerant Systems*, pages 121–140. Springer-Verlag, Vienna, Austria, 1987.

[5] M. McElvany Hugue. Estimating reliability under the static hybrid fault model. Technical report, Aerospace Technology Center, Allied-Signal Aerospace Company, Columbia, MD, 1992.

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

[7] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine Generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, July 1982.

[8] Patrick Lincoln and John Rushby. Formal verification of an algorithm for interactive consistency under a hybrid fault model. Technical Report SRI-CSL-93-2, Computer Science Laboratory, SRI International, Menlo Park, CA, March 1993. Also available as NASA Contractor Report 4527, July 1993.

[9] Patrick Lincoln and John Rushby. Formal verification of an algorithm for interactive consistency under a hybrid fault model. In Costas Courcoubetis, editor, *Computer-Aided Verification, CAV '93*, volume 697 of *Lecture Notes in Computer Science*, pages 292–304, Elounda, Greece, June/July 1993. Springer-Verlag.

[10] Dale A. Mackall. Development and flight test experiences with a flight-crucial digital control system. NASA Technical Paper 2857, NASA Ames Research Center, Dryden Flight Research Facility, Edwards, CA, 1988.

[11] Fred J. Meyer and Dhiraj K. Pradhan. Consensus with dual failure modes. In *Fault Tolerant Computing Symposium 17*, pages 48–54, Pittsburgh, PA, July 1987. IEEE Computer Society.

[12] Fred J. Meyer and Dhiraj K. Pradhan. Consensus with dual failure modes. *IEEE Transactions on Parallel and Distributed Systems*, 2(2):214–222, April 1991.

[13] S. Owre, J. M. Rushby, and N. Shankar. PVS: A prototype verification system. In Deepak Kapur, editor, *11th International Conference on Automated Deduction (CADE)*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 748–752, Saratoga, NY, June 1992. Springer-Verlag.

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

[15] John Rushby. Formal verification of an Oral Messages algorithm for interactive consistency. Technical Report SRI-CSL-92-1, Computer Science Laboratory, SRI International, Menlo Park, CA, July 1992. Also available as NASA Contractor Report 189704, October 1992.

[16] John Rushby and Friedrich von Henke. Formal verification of the Interactive Convergence clock synchronization algorithm using EHDM. Technical Report SRI-CSL-89-3R, Computer Science Laboratory, SRI International, Menlo Park, CA, February 1989 (Revised August 1991). Original version also available as NASA Contractor Report 4239, June 1989.

[17] John Rushby and Friedrich von Henke. Formal verification of algorithms for critical systems. In *SIGSOFT '91: Software for Critical Systems*, pages 1–15, New Orleans, LA, December 1991. Expanded version to appear in *IEEE Transactions on Software Engineering*, 1993.

[18] Natarajan Shankar. 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, Nijmegen, The Netherlands, January 1992. Springer-Verlag.

[19] Philip Thambidurai and You-Keun Park. Interactive consistency with multiple failure modes. In *7th Symposium on Reliable Distributed Systems*, pages 93–100, Columbus, OH, October 1988. IEEE Computer Society.

[20] Philip Thambidurai, You-Keun Park, and Kishor S. Trivedi. On reliability modeling of fault-tolerant distributed systems. In *9th International Conference on Distributed Computing Systems*, pages 136–142, Newport Beach, CA, June 1989. IEEE Computer Society.

[21] John H. Wensley, Leslie Lamport, Jack Goldberg, Milton W. Green, Karl N. Levitt, P. M. Melliar-Smith, Robert E. Shostak, and Charles B. Weinstock. SIFT: Design and analysis of a fault-tolerant computer for aircraft control. *Proceedings of the IEEE*, 66(10):1240–1255, October 1978.

Appendix A: Specification in PVS

The formal specification imports some predefined theories and defines some shorthand notations for treating fcu (channel) statuses as predicates and filters. There are a few lemmas easily proved by induction about cardinality, filters, and remove. Then *send* is axiomatized.

send is a function that captures the properties of sending values from one channel to another. We axiomatize *send* as a function that takes a value to be sent, a sender, and a receiver as arguments, and returns the result received by the receiver. A drawback to this formal specification of send is that even arbitrarily faulty processors must always send the same (possibly bad) value to the same processor, even in different rounds of the protocol. This fact is not exploited in the proof, but it may be preferable to develop a relational axiomatization that does not make this assumption. Natarajan Shankar has axiomatized the OM algorithm using a relational send that avoids this assumption, and has proven the resulting correctness conditions as an exercise. *distr* is shorthand for a curried *send*. *HybridMajority* is axiomatized in a way that facilitates its application to the proof of *Validity*. Essentially, it says that the hybrid majority of a set is the correct value, if all good processors in the set agree on that correct value, which is not E, all crashed processors appear to have value E, and there are enough good processors to outvote the remaining Byzantine and value faulty processors. R and *UnR* are axiomatized, and finally we specify the algorithm *OMH*.

OMH is specified in a higher-order style that facilitates the PVS proof, but which is a somewhat different viewpoint from the informal description given above. *OMH* takes four arguments: G, the Transmitter, r the number of rounds, $v1$, a vector of values sent by the Transmitter, and *caucus*, the set of processors participating in this invocation of *OMH*. In the case that there are no rounds of message passing, that is, $r = 0$, *OMH* simply reduces to the vector of values sent by the Transmitter as $v1$. Otherwise, *OMH* is the vector described by the lambda-expression. That is, given a processor p, if p is the Transmitter, then just use the value the Transmitter sent himself. If p is not the Transmitter, then we apply *UnR* to the hybrid majority of a set of values. These value are arrived at by recursive calls to *OMH* where each processor in turn assumes the role of Transmitter, sending the R of its value to all other processors in the caucus (including itself).

Validity is one of the two key lemmas, stating that if there are enough good processors, and the Transmitter is not Byzantine faulty, then running *OMH* is the same as simply sending from the Transmitter to the final receiver. Agreement is the other key lemma, stating that if there are enough good processors, then no matter what state the Transmitter is in, all good processors eventually agree on the final result. Finally, *ValidityFinal* and *AgreementFinal* are stated as the final theorems of interest.

The CrashOnly variants of these properties embody the formal specification of Theorem 2, showing that the algorithm is optimal in the case that there are no arbitrary nor value faults. Note that analogous ArbitraryOnly theorems also hold, giving optimal bounds, although these bounds are trivial consequences of the general theorem. Also note that analogous ValueOnly theorems would not give optimal bounds.

```
omh[m : nat, n : posnat,  T : TYPE,  error : T,   R,  UnR : [T → T]] :
THEORY BEGIN
ASSUMING  act :  ASSUMPTION (∀ (t : T) :  R(t) ≠ error)
          unact :  ASSUMPTION (∀ (t : T) :  UnR(R(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 finite_cardinality[fcu, n,  identity[fcu]],
          filters[fcu],
          card_set[fcu, n,  identity[fcu]],
           hybrid_mjrty[T, n,  error]

statuses :  TYPE  = {arbitrary, symmetric, manifest, nonfaulty}
status :  [fcu → statuses]
a(z) :  bool  = arbitrary(status(z))
s(z) :  bool  = symmetric(status(z))
c(z) :  bool  = manifest(status(z))
g(z) :  bool  = nonfaulty(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)

send :  [T,  fcu,  fcu → T]
send1 :  AXIOM g(p) ⊃ send(t, p, q) = t
send2 :  AXIOM c(p) ⊃ send(t, p, q) = error
send3 :  AXIOM s(p) ⊃ send(t, p, q) = send(t, p, z)
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 UnR(HMajority(caucus − {G},
                              (λ q : OMH(q, r − 1, R(send(t, G, q)),
               ENDIF)
   ENDIF
 MEASURE (λ G, r, t, caucus → nat :  r)

Validity : THEOREM
 g(p) ∧ ¬ a(G)  ∧ 2 × |a| + 2 × |s| + |c| + m < n
   ⊃ OMH(G, m, t,  fullset[fcu])(p) = send(t, G, p)

Agreement : THEOREM
 g(p) ∧ g(q) ∧ |a| ≤ m ∧ 2 × |a| + 2 × |s| + |c| + m < n
   ⊃ OMH(G, m, t,  fullset[fcu])(p) = OMH(G, m, t,  fullset[fcu])(q)
 END omh
```