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

Preprint of a paper to be presented at the Fault-Tolerant Computing Symposium, FTCS 23, Toulouse, France, June 1993

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

Abstract

Thambidurai and Park [13] have proposed an algorithm for Interactive Consistency that retains resilience to the arbitrary (or Byzantine) fault mode, while tolerating more faults of simpler kinds than standard Byzantine-resilent algorithms. Unfortunately, and despite a published proof of correctness, their algorithm is flawed. We detected this while undertaking a formal verification of the algorithm.

We present a corrected algorithm that has been subjected to mechanically-checked formal verification. Because informal proofs seem unreliable in this domain, and the consequences of failure could be catastrophic, we believe formal verification should become standard for algorithms intended for safety-critical applications.

Keywords: Byzantine agreement, interactive consistency, 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

Byzantine-resilient algorithms make no assumptions about the behavior of faulty components and are therefore maximally effective with respect to the ***kinds*** (or ***modes***) of faults they tolerate. But they are not uniformly effective with respect to the ***number*** of faults they can tolerate: other algorithms can withstand more faults for a given level of redundancy than Byzantine-resilient ones, provided the faults are of particular kinds. However, these alternative algorithms may fail when confronted by faults beyond the kinds they are designed to handle.

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 those 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 achieve 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.[1]

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 [10] in 1980.

The principal difficulty to be overcome in achieving Interactive Consistency is the possibility of 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. Interactive Consistency algorithms overcome this problem 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

[1] There are examples where unanticipated behaviors of the mechanisms for fault tolerance became the *primary* source of system failure [6].

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.[2]

Using $m+1$ rounds of message exchanges, the Oral Messages algorithm of Lamport, Shostak, and Pease [3], which we denote 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 [10]. However, as we have already noted, OM(m) is not optimal when other than arbitrary faults are considered: other 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 [13] and Meyer and Pradhan [7] have considered Interactive Consistency algorithms that resist multiple fault classes. Thambidurai and Park's "unified" or "hybrid" fault model divides faults into three classes: ***nonmalicious*** (or ***benign***), ***symmetric malicious***, and ***asymmetric malicious***. We find the anthropomorphism in terms such as "malicious faults" unhelpful and rename the cases to ***arbitrary***, ***symmetric***, and ***manifest*** faults, respectively. A manifest fault is one that produces detectably missing values (e.g., timing, omission, or crash faults), or that produces a value that all nonfaulty recipients can detect as bad (e.g., it fails checksum or format tests). The other two fault modes yield values that are not detectably bad (i.e., they are ***wrong***, rather than missing or manifestly corrupted, values): a symmetric fault delivers the same wrong value to every nonfaulty receiver; an arbitrary fault is completely unconstrained and may deliver (possibly) different wrong values (or missing or detectably bad values) to different nonfaulty receivers.

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.[3]

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" of incorrect algorithms for this problem ourselves. We discovered the errors in Thambidurai and Park's algorithm and in our own imperfect variants while attempting to formally verify the algorithms concerned.

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

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

The algorithm presented here has been subjected to mechanically-checked formal verification using the PVS verification system [8]. We describe this formal verification and claim that it is not particularly difficult. Because informal proofs seem unreliable in this domain, and because the consequences of failure could be catastrophic, we argue that formal verification should become standard.

2 Requirements, Assumptions, and the Algorithms OM and Z

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 [3] 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. All the algorithms we consider are presented here in their Byzantine Generals formulation. 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 [11] for more extended discussion of this topic.

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*. 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, $\nu_p = v$.

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.

Next, we 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 (i.e., that it achieves BG1 and BG2 under assumptions A1 to A3) and its optimality (i.e., that no algorithm can mask the same number of arbitrary faults with fewer processors) were proven in [3, page 390]. These results have been formally verified by Bevier and Young [1].

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 (we call this a ***hybrid-majority*** vote). Thambidurai and Park claim that an m-round implementation of Algorithm Z can withstand $a+s+c$ simultaneous faults, where a is the number of arbitrary faults, s the number of symmetric faults, and c the number of manifest faults,[4] provided $a \leq m$, and n, the number of processors, satisfies $n > 2a+2s+c+m$. In the case of no symmetric or manifest faults (i.e., Byzantine faults only), we have $m = a$ and $s = c = 0$, 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 [11], and have identified deficiencies in some of the previously published analyses (though not in

[4]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.

the algorithms) [9, 12]. 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 [13, pages 96 and 97]. This proof follows the outline of the standard proof for OM [3, page 390] quite closely. However, we found that Thambidurai and Park's proof of 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 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 manifest fault, one of the receivers has an arbitrary fault, and the algorithm is Z with one round (i.e., $n = 5, a = 1, s = 0, c = 1, m = 1$). 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.

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

As noted, the fault modes we distinguish for processors are ***arbitrary-faulty***, ***symmetric-faulty***, and ***manifest-faulty***. Of course, we also need a class of ***good*** (also called ***nonfaulty***) processors. We specify these fault modes semiformally as follows.

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[5]
- 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.

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

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

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.

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.

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

[5]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.

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 therefore 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 for the time being that the functions R and UnR are applied to *all* values

(alternatives are explored in Section 5). This gives 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.

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-E values v_q received, (i.e., the ***hybrid-majority***); if no such

majority exists, the receiver uses some arbitrary, but functionally determined value. Receiver p then applies UnR to that value, using the result as the transmitter's value.

3.3 Correctness Arguments

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.)

The argument for the correctness of OMH is an adaptation of that for the Byzantine Generals formulation of OM [3, 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,
- 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.

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, $c' \leq c$, and $1 \leq m$, 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 $m \geq a$.*

Proof: The proof is by induction on m. In the base case $m = 0$ there can be no arbitrary-faulty processors, since $m \geq a$. 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 $2(a + s) + c + m$ 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. □

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 ($n > 3a$) and at least as many rounds as arbitrary-faulty processors ($m \geq a$). 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 (i.e., if $s = c = 0$). However, OMH achieves can withstand larger numbers of symmetric and manifest faults than OM—which does not distinguish such faults from arbitrary ones.

However, even OMH appears suboptimal in the number of faults tolerated in some of the extreme circumstances. In some cases, this is because algorithm is truly suboptimal; in others, the algorithm is optimal, but the general analysis given above is too conservative. As an example of the latter, consider the case where only manifest faults are present. Then the general analysis above indicates that the number of manifest faults that can be tolerated is $n-m-1$: in other words, the greater the number of rounds, the ***fewer*** manifest faults that can be tolerated. In fact, alternative analysis shows that OMH(m) tolerates the maximum possible number of manifest-faulty processors when there are no arbitrary nor symmetric faults. The only constraint is that there must be more processors (whether faulty or not) than rounds (since otherwise some recursive instances would be run on the empty set of processors).

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

This theorem has been formalized and mechanically verified [4].

When only symmetric faults are present, it is the algorithm, rather than its general analysis, that is less than optimal. Here, the additional rounds of message exchanges are actively counterproductive in the cases $m > 0$ (compare $n = 4$, $s = 2$ 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 symmetric faults. An interesting topic for future research is to investigate whether this trade-off can be mitigated.

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 Ability of OMH(1) with $n = 6$

Table 1 summarizes the different numbers of simultaneous faults that a 6-plex can withstand using OMH(1); for comparison, observe that the standard analysis indicates that OM(1) can withstand only a single (arbitrary) fault in this configuration. Thambidurai, Park and Trivedi [14] present reliability analyses that show this increased fault-tolerance indeed provides superior reliability under plausible assumptions[6].

In fact, OM(1) can itself withstand more faults than its standard analysis suggests. When there are no manifest faults, Algorithm OMH becomes similar to the traditional Algorithm OM. A related point was made in [13]: in the absence of error values, hybrid majority is equivalent to majority. Thus the only substantive difference between OMH and OM are the wrapper and unwrapper functions applied to values. As discussed in Section 5 these functions may be identity on nonerror values, in which case OMH becomes exactly OM. Thus the analysis of the previous section may be applied, showing that the traditional algorithm OM(m) satisfies conditions BGH1 and BGH2 if there are more than $2(a + s) + m$ processors and $m \geq a$. Thus, the real distinction between OM and OMH is that the latter can distinguish manifest from (other) symmetric faults. In the case tabulated above ($n = 6, m = 1$), this revised analysis means that OM can withstand two simultaneous faults, provided at most one of them is arbitrary.

4 Formal Verification of OMH

We have formally verified Lemma 1 and Theorems 1 and 2 for OMH(n) using the PVS verification system [8]. That is, we have expressed the algorithm, its assumptions, and desired properties in the formal specification language of PVS, and have developed formal proofs of the desired properties that have been accepted by the PVS proof checker. Technical descriptions of the formal specification and verification are given in a companion paper [5], and presented in complete detail in a

[6] Although algorithm Z is somewhat flawed, the analysis in [14] can be correctly applied to OMH

technical report [4]; here we focus on how we performed the formal verification and on the benefits we derived.

The specification language of PVS is a higher-order logic with a rather rich type system in which it is comparatively straightforward to state the desired specification. Although OMH is conceived as a distributed, concurrent algorithm, its correctness argument need not involve a model of distributed computation and we were therefore able to specify OMH as a simple recursive function. Such abstraction is one of the keys to making formal analysis of difficult algorithms tractable. The formal specification of OMH was derived from one we had previously constructed for the classical OM algorithm [11] and was developed iteratively as failed attempts at formal verification exposed the errors described earlier in Algorithm Z and its plausible variants.

Ideally, formal verification should resemble a dialog with a tirelessly skeptical colleague who checks every detail of a purported proof and makes sure that no cases are overlooked. The proof checker of PVS falls a long way short of this ideal, but comes closer to it than others. In contrast to more automatic theorem provers, which must be coaxed to "discover" the proof themselves, the user of PVS proposes the main proof steps directly (e.g., "use induction on formula 3," or "consider the case $m = 0$") and PVS carries them out. The primitive inference steps of PVS are quite powerful and include decision procedures for linear arithmetic, so that trivial steps such as $m + 1 - 1 = m$ are dealt with instantaneously. Less powerfully automated proof checkers require user assistance to discharge the hundreds of such trivial facts that arise in a proof; this considerably slows the development of a proof and, more importantly, distracts the user from the main line of the argument.

The greatest benefit of formal specification and verification in this case has been the refinement of our own understanding. It is very easy for humans to be convinced of the correctness of flawed algorithms in domains where lots of detail and special cases must be considered. In more than one case during the development of OMH, we developed convincing informal arguments and attempted to verify the claims using PVS. The proof checker would not accept these flawed arguments, and eventually led us to discover counterexamples. Finally, we were able to develop the new algorithm presented above, and prove it correct.

It is sometimes argued that a large part of the value of formal methods lies in formal specification rather than verification. Formal specifications can serve to clarify thinking and also provide a means of communication less subject to error and misinterpretation than traditional natural-language documentation. However, the additional step of verification proved crucial in our case. Formal specification of the flawed algorithms strengthened our erroneous convictions about them; only through failed attempts at formal verification were the errors detected.

Even formal verification is insufficient to guarantee that an algorithm is fit for its intended purpose: *validation* is required as well, in order to ensure that the assumptions are realistic, and that the properties established match those required for the

intended application. Peer review is an essential element in validation, but another technique we use is to pose "challenges" that are evaluated by theorem proving. For example, an axiomatization of the hybrid majority function can be challenged by proving that the hybrid majority of a collection of values not containing E is the same as the simple majority of that collection. In our case, we produced one specification of a flawed algorithm for which we were able to formally verify the validity property. However, one of our axioms about hybrid majority was stated incorrectly. Only through challenges and attempting to refine the specification into an implementation did the inadequacy of the axiom become apparent.

5 Implementing R and UnR

Although our presentation of the OMH Algorithm suggests that R and UnR are applied to all values at every round, this is unnecessary. R and UnR may be identity on nonerror values. Thus, values v could be passed with an extra (say, highest order) bit denoting whether the word actually stands for a data value or for $R^v(E)$. R and UnR would then become increment and decrement operations conditional on the highest bit.

If R and UnR are applied to all values at every round, perhaps as unconditional increment and decrement operations, then intermediate error values such as $R(R(E))$ may coincide with valid data values. The algorithm remains correct because UnR (decrement) is always applied to the output of the majority vote.

Both of these implementations of R and UnR require unbounded integers in order to truly satisfy the requirements on R and UnR (for all v, $R(v) \neq E$, and $UnR(R(v)) = v$). However, for an m round OMH, just $m+1$ error values (E up to $R^m(E)$) suffice with suitable modifications to the algorithm.

One could add a comparison of the number of applications of R with the depth of recursion in the algorithm OMH. (Simply computing $R^x(E)$ where x is taken modulo the total number of rounds leads to erroneous results.) Any values with more R's than elapsed rounds may correctly be considered to indicate manifest faults and treated as E, thus reducing the number of possible error values to one more than the number of rounds. In the common case of one-round OMH, two error values, corresponding to E and $R(E)$ suffice. With only a small set of error values, it may no longer be necessary to distinguish them by setting a special bit: they could simply be allocated to values beyond the valid data range.

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. These implementation techniques have not been formally verified.

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 flawed. In the preceding sections, we have described the problem with Algorithm Z and presented OMH, a correct algorithm for this problem.

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 model of the program is shown to satisfy a specification, and shown to exhibit certain properties under a certain set of assumptions. 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 tireless mechanical skeptic.

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 [8]. 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 [9, 12], we suggest that formal verification should become a routine part of the social process of development and analysis for fault-tolerant algorithms intended for practical application in safety-critical systems.

Acknowledgments: PVS was constructed by our colleagues Sam Owre and Natarajan Shankar. Michelle McElvany-Hugue and Chris Walter of Allied Signal provided helpful discussion on hybrid fault models. The anonymous referees provided very useful comments.

References

[1] William R. Bevier and William D. Young. Machine checked proofs of the design of a fault-tolerant circuit. *Formal Aspects of Computing*, 4(6A):755–775, 1992.

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

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

[4] Patrick Lincoln and John Rushby. Formal verification of an algorithm for interactive consistency under a hybrid fault model. Technical Report SRI-CSL-93-02, Computer Science Laboratory, SRI International, Menlo Park, CA, March 1993.

[5] Patrick Lincoln and John Rushby. Formal verification of an algorithm for interactive consistency under a hybrid fault model. In *Computer-Aided Verification*, Crete, July 1993. Accepted for publication.

[6] 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.

[7] 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.

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

[9] Sam Owre, John Rushby, Natarajan Shankar, and Friedrich von Henke. Formal verification for fault-tolerant architectures: Some lessons learned. In *FME '93: Industrial-Strength Formal Methods*, pages 482–500, Odense, Denmark, April 1993. Volume 670 of *Lecture Notes in Computer Science*, Springer Verlag.

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

[11] 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.

[12] John Rushby and Friedrich von Henke. Formal verification of algorithms for critical systems. *IEEE Transactions on Software Engineering*, 19(1):13–23, January 1993.

[13] 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.

[14] 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.