A Formally Verified Algorithm for Clock Synchronization Under a Hybrid Fault Model*

Reprint from the 13th ACM Symposium on Principles of Distributed Computing (PODC '94), pp. 304–313, Los Angeles CA, August 1994

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

Abstract

A small modification to the interactive convergence clock synchronization algorithm allows it to tolerate a larger number of simple faults than the standard algorithm, without reducing its ability to tolerate arbitrary or "Byzantine" faults. Because the extended case-analysis required by the new fault model complicates the already intricate argument for correctness of the algorithm, it has been subjected to mechanically-checked formal verification.

The fault model examined is similar to the "hybrid" one previously used for the problem of distributed consensus: in addition to *arbitrary* faults, we also admit *symmetric* (i.e., consistent) and *manifest* (i.e., detectable) faults. With n processors, the modified algorithm can withstand a arbitrary, s symmetric, and m manifest faults simultaneously, provided $n > 3a + 2s + m$. A further extension to the fault model includes *link* faults with bound $n > 3a + 2s + m + l$ where l is the maximum, over all pairs of processors, of the number of processors that have faulty links to one or other of the pair.

The mechanically-checked formal verification of the modified algorithm was achieved by extending one for the classical Interactive Convergence algorithm, and was accomplished relatively easily. A mechanically-checked formal specification and verification is a reusable intellectual resource whose initial cost is amply repaid by the support it provides for inexpensive and reliable investigation of modified assumptions and algorithms such as those reported here.

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

1 Introduction

Fault-tolerant architectures for digital flight control in civil aircraft provide the context for the work described here. These architectures generally use relatively few processors (from four to nine) and a topic of major practical concern is the development of fault-tolerant algorithms that can withstand as many *kinds* and as large a *number* of faults as possible within a given level of redundancy. Byzantine-fault-tolerant algorithms are attractive in this context because they can withstand any kind of fault, but they require a great deal of redundancy and can be overwhelmed by relatively few faults, even if the faults are of a simple kind that could be tolerated by other algorithms.

An interesting line of investigation, therefore, is to seek algorithms that can tolerate faults of many different kinds, and that "use up" redundancy according to the severity of the faults that actually arrive. Among the earliest algorithms of this type was one by Perry and Toueg for distributed consensus under combinations of processor and communication faults [14]. Extending such algorithms to tolerate truly arbitrary (Byzantine) faults is difficult because the *possibility* that a fault may display asymmetric symptoms complicates the treatment of all faults. A complementary approach is to start with Byzantine-fault-tolerant algorithms and then try to improve their treatment of simpler fault modes. Meyer and Pradhan [10] and Garay and Perry [2] have developed fault models and algorithms of this kind by adding crash faults to the classical Byzantine model. Thambidurai and Park [20] propose a more elaborate *hybrid* fault model: they widen the notion of crash faults to include any fault that produces *manifest* (i.e., detectable) symptoms, and they allow *symmetric* faults (i.e., those that exhibit incorrect but consistent behavior) in addition to Byzantine faults. Thambidurai and Park present a modification to the Oral Messages algorithm of Lamport, Pease and Shostak [6] that achieves consensus under their fault model; this algorithm is employed in the MAFT ("Multicomputer Architecture for Fault Tolerance") system for flight-control applications [4]. Unfortunately, the algorithm and its proof of correctness are flawed (though its implementation in MAFT is not). The flaw was detected through a failed attempt at formal verification by Lincoln and Rushby, who then developed a corrected algorithm [8], and a mechanically-checked formal verification of its correctness [7].

All the works cited above deal with consensus; clock synchronization in the presence of multiple fault modes has received less attention. Infis and Moore [3] argue that the incidence of truly Byzantine (i.e., asymmetric) faults can be reduced to a very low level by suitable hardware design, and they present a synchronization algorithm that withstands t "all-but-Byzantine" faults with only $2t+1$ processors (Byzantine fault-tolerant synchronization requires $3t+1$ processors). Garay and Perry [2] state that they have algorithms for clock synchronization under their "Byzantine plus crashes" fault model, but they do not describe them.

In this paper, I consider the problem of clock synchronization under a fault model similar to that introduced for consensus by Thambidurai and Park. I show that a slightly modified version of the classical Interactive Convergence algorithm of Lamport and Melliar-Smith is able to tolerate a arbitrary, s symmetric, and m manifest faults simultaneously, provided $n > 3a + 2s + m$. I also present an extension to the fault model that admits a type of (intermittent) link fault and show that these are no more expensive to tolerate than manifest faults.

Algorithms such as this are intended to support safety-critical applications, so strong assurance is required for their correctness. Unfortunately, there is evidence that the traditional processes of informal proof and peer review do not always provide this assurance reliably. For example, as noted above, the first published algorithm for consensus under a hybrid fault model was incorrect; the error was detected and corrected with the aid of mechanically-checked formal verification. In other cases, the algorithms have been correct, but the proofs that support that claim have been flawed, and some of the underlying assumptions have been incompletely or improperly characterized. For example, the interactive convergence clock synchronization algorithm of Lamport and Melliar-Smith, which provides the foundation for the algorithm examined here, was published with a proof of correctness supported by five lemmas [5]. Formal verification of this argument by Rushby and von Henke revealed that the proofs of the main theorem and of all but one of the lemmas are flawed [15, 16]. Numerous Byzantine-fault-tolerant clock synchronization algorithms have appeared since those of Lamport and Melliar-Smith; Schneider presents a uniform treatment that includes almost all the known algorithms [18]. Shankar constructed a mechanically-checked formal verification of Schneider's argument and, again, several small errors were detected [19]. (Schneider and Shankar are preparing a corrected treatment for publication.)

These deficiencies in proofs of distributed algorithms for consensus and clock synchronization were all discovered and corrected with the aid of mechanized verification systems. The proofs concerned are not mathematically deep, but they involve rather intricate arguments, considerable case analysis, and many delicate assumptions—characteristics that particularly favor the painstaking bookkeeping of mechanized verification.

The algorithm presented here has been subjected to mechanized verification using the EHDM system.[1] Because the algorithm is a variation on Interactive Convergence—which had already been analyzed using EHDM [15, 16]—development of its formal specification and verification was not particularly difficult or time-consuming. The support provided for reliable and inexpensive investigation of modifications to algorithms is an inadequately recognized benefit of formal verification. In the present case, formal verification was instrumental in developing an algorithm for clock synchronization which, among all those suitable for architectures of the kind used in digital flight-control applications, seems to provide the most robust fault tolerance for a given level of redundancy.

The fault model, the modified algorithm, and its analysis are presented in the next section. The mechanically-checked formal verification of the algorithm is described in Section 3. Conclusions are presented in the final section.

2 The Algorithm and Its Analysis

The starting point for the algorithm developed here is the Interactive Convergence Algorithm (ICA) of Lamport and Melliar-Smith. To indicate the modifications made to accommodate a hybrid fault model, it is necessary to present a few elements from the formal treatment of the classical algorithm; for brevity, I omit many details (including the definitions of good clocks and nonfaulty processors). The notation, terminology, and the naming and numbering of assumptions and lemmas are those introduced by Lamport and Melliar-Smith [5]; however the statements of assumptions, lemmas, and

[1] EHDM [17] is a verification system constructed at SRI whose distribution is controlled by the US Government. PVS [12] is a system of similar capabilities that is freely available from SRI—by anonymous ftp from `ftp.csl.sri.com/pub/pvs` or by World Wide Web from `http://www.csl.sri.com/sri-csl-pvs.html`.

constraints are from our corrected treatment [15, 16] which, among other changes, eliminates the approximations used by Lamport and Melliar-Smith.

2.1 The Interactive Convergence Algorithm

Two notions of time are distinguished: *clock time* is the internal estimate of time that a processor obtains from its clock, while *real time* is an external, abstract notion of time that provides a common frame of reference. Upper case Latin and Greek letters are used for clock time quantities, lower case for real time.

The goal of ICA is to maintain the clocks of redundant processors within some bounded skew δ of each other: that is to say, the real time difference between the instant when the clock of processor p reads T and that when the clock of processor q reads T must be less than δ. All processors have reasonably accurate clocks with a maximum rate of drift from real time given by ρ, and are synchronized within some bound δ_0 initially. Each processor engages in the synchronization protocol every R seconds, and for a duration of S seconds, according to its own clock. During synchronization, each processor determines the differences between its own clock and those of other processors, forms a "fault-tolerant average" of those differences, and adjusts its own clock by that amount. Processors may not be able to determine the differences between their clocks with absolute accuracy: the quantity ϵ bounds the error in $\Delta_{qp}^{(i)}$, which denotes the difference between the clocks of processors q and p, as determined by p during its ith resynchronization. The key to making a clock synchronization algorithm resistant to Byzantine failures among its components is the use of a "fault-tolerant average" in the adjustment step [18]. ICA is characterized by use of the *egocentric mean* as its fault-tolerant averaging function. To compute the egocentric mean, a processor replaces all differences $\Delta_{qp}^{(i)}$ larger than a fixed quantity Δ by zero, and then calculates the arithmetic mean of the resulting set of differences.

To formalize this description, clocks are modeled as functions from clock to real time: $c_p(T)$ denotes the real time at which processor p's physical clock reads T. Clocks are adjusted by subtracting a correction from their readings; the correction for processor p in the ith R-second *period* is denoted $C_p^{(i)}$, and

$$c_p^{(i)}(T) = c_p(T + C_p^{(i)})$$

is defined as the *logical* clock for p during period i. The ith period is denoted $R^{(i)}$ and runs from clock time $T^{(i)}$ to $T^{(i+1)}$, where $T^{(i)} = T^0 + iR$ $(i \geq 0)$ and T^0 is an arbitrary constant. Synchronization takes place during the last S seconds of each period; $S^{(i)}$ denotes the interval $[T^{(i+1)} - S, T^{(i+1)}]$.

ICA is required to maintain two conditions called S1 and S2. The first says that the skew between the clocks of nonfaulty processors must be bounded (i.e., the algorithm must maintain synchronization).

Clock Synchronization Condition S1: *For all processors p and q, if all but at most t processors (out of n) are nonfaulty through period i and if p and q are nonfaulty through period i, then for all T in $R^{(i)}$*

$$|c_p^{(i)}(T) - c_q^{(i)}(T)| < \delta.$$

The other condition, S2, which is ignored here, says that corrections must be bounded (this is needed to eliminate trivial solutions).

ICA requires that the processors start off with their clocks approximately synchronized, and that each nonfaulty processor can read the difference between its own clock and that of another nonfaulty processor with a bounded error. Any implementation of the algorithm must satisfy these two assumptions which, for historical reasons, are called A0 and A2. For brevity, A0 is ignored here. Note that in order to read clock differences, the processors may already need to be synchronized, so S1 and S2 are required to hold in A2.

Assumption A2: *If conditions S1 and S2 hold for the i'th period, and processor p is nonfaulty through period i, then for each other processor q, p obtains a value $\Delta_{qp}^{(i)}$ during the synchronization period $S^{(i)}$. If $p = q$, then $\Delta_{qp}^{(i)} = 0$; otherwise, if q is also nonfaulty through period i, then $|\Delta_{qp}^{(i)}| \leq S$ and*

$$|c_p^{(i)}(T' + \Delta_{qp}^{(i)}) - c_q^{(i)}(T')| < \epsilon$$

for some time T' in $S^{(i)}$. (The value is unconstrained if q is not nonfaulty through period i.)

The Interactive Convergence Algorithm is specified as follows.

Algorithm ICA: *For all processors p:*

$$C_p^{(i+1)} = C_p^{(i)} + \Delta_p^{(i)},$$

where

$$\begin{aligned} \Delta_p^{(i)} &= \left(\frac{1}{n}\right) \sum_{r=1}^{n} \bar{\Delta}_{rp}^{(i)}, \quad \textbf{and} \\ \bar{\Delta}_{rp}^{(i)} &= \textbf{if } |\Delta_{rp}^{(i)}| < \Delta \textbf{ then } \Delta_{rp}^{(i)} \textbf{ else } 0. \end{aligned}$$

Thus, the correction $\Delta_p^{(i)}$ is the egocentric mean of the clock differences $\Delta_{rp}^{(i)}$, where Δ is a parameter to the algorithm that determines when a clock difference is "too large."

The proof that ICA achieves S1 and S2, subject to A0 and A2, depends on 5 lemmas and on seven constraints on its parameters. The most interesting of the Lemmas are numbers 4 and 5, which consider the contribution to the skew between nonfaulty processors p and q due to the differences they observe between their own clocks and that of processor r. Lemma 4 considers the case where r is nonfaulty, Lemma 5 the case where it is faulty.

Lemma 4. *If the clock synchronization conditions S1 and S2 hold for i, processors p, q, and r are nonfaulty through period $i+1$, and $T \in S^{(i)}$, then*

$$|c_p^{(i)}(T) + \bar{\Delta}_{r\,p}^{(i)} - [c_q^{(i)}(T) + \bar{\Delta}_{r\,q}^{(i)}]| < 2(\epsilon + \rho S) + \rho\Delta.$$

Lemma 5. *If the clock synchronization conditions S1 and S2 hold for i, processors p and q are nonfaulty through period $i+1$, and $T \in S^{(i)}$, then*

$$|c_p^{(i)}(T) + \bar{\Delta}_{r\,p}^{(i)} - [c_q^{(i)}(T) + \bar{\Delta}_{r\,q}^{(i)}]| < \delta + 2\Delta.$$

If there are n processors, of which t are faulty, the skew between nonfaulty processors p and q in the $i+1$st period is the mean of $n-t$ terms derived from Lemma 4 and t terms derived from Lemma 5, plus some correction terms. More exactly, it is

$$\left(\frac{1}{n}\right)[(n-t)(2[\epsilon + \rho S] + \rho\Delta) + t(\delta + 2\Delta)] + \rho(R + \Sigma).$$

To maintain synchronization, this expression must be no greater than δ. Rearranging terms, this gives

Constraint C6:

$$\delta \geq 2(\epsilon + \rho[S + \frac{\Delta}{2}]) + \frac{2t\Delta}{n-t} + \frac{n\rho(R+\Sigma)}{n-t},$$

which is the bound on the synchronization achieved.

As noted, the proof that ICA achieves S1 and S2 subject to A0 and A2, requires that seven constraints, named C0 to C6, on its parameters are satisfied. We have already seen C6; another is

Constraint C4:

$$\Delta \geq \delta + \epsilon + \frac{\rho}{2}S.$$

Synchronization can be achieved only if all seven constraints are satisfied simultaneously. However, taking C4 with C6, it can be seen that a necessary condition is $n > 3t$.

2.2 The Hybrid Fault Model and the Modified Algorithm

Thambidurai and Park's fault model for distributed consensus distinguishes three fault modes: *manifest* faults are those that can be detected by every nonfaulty processor; *symmetric* faults deliver wrong rather than detectably bad values, but all nonfaulty processors receive the same values; *arbitrary* faults are completely unconstrained. Because clock synchronization is a different problem than consensus, we have to reinterpret this fault model for the present problem, while seeking to retain the plausibility of its motivation. For clock synchronization, communication between processors is limited to determination of the clock differences $\Delta_{q\,p}^{(i)}$, so the fault model needs to be expressed in these terms.

In previous work extending the Oral Messages algorithm for consensus [8], it was found that symmetric faults were already dealt with, fortuitously, by the ordinary algorithm—though additional analysis is required to reveal that fact—and that the key to operating under a hybrid fault model is to diagnose and deal specially with manifest faults. The same is true for ICA: we have to find a definition for symmetric faults that places them within the competency of the ordinary algorithm, and a suitable special treatment for manifest faults. The special treatment that is suitable for ICA, as it is for Oral Messages, is simply to exclude manifest-faulty values from further consideration. Since ICA operates by reading clock differences and averaging them, we can do this by setting any manifestly faulty readings of clock differences to zero. Although this is really an adjustment to the algorithm, it is more convenient to formalize it as an additional clause in its Assumption A2—though it must be realized that the implementation of the reading of clock differences must be specifically designed to satisfy this assumption. Methods of detecting manifest faults may include timeouts, checksums, or other similar tests.

For symmetric faults, we want to capture the behavior of processors that are doing the wrong thing, but are doing it consistently. For consensus, this is specified by stating that each nonfaulty receiver gets the same value—though it may not be the right value. In clock synchronization, there is some uncertainty in reading clock differences, so we cannot require them to be the same for different receivers: we can, however, require them to be "close" together—that is require $|\Delta_{q\,p}^{(i)} - \Delta_{q\,r}^{(i)}| < \Gamma$ for some suitable parameter Γ. It turns out that Δ is the largest useful value for Γ. I formalize these interpretations of manifest and symmetric faults in the following restatement of Assumption A2.

Assumption A2 (for Hybrid Fault Model): *If conditions S1 and S2 hold for the i'th period, and processor p is nonfaulty through period i, then for each other processor q, p obtains a value $\Delta_{qp}^{(i)}$ during the synchronization period $S^{(i)}$ such that:*

- *If $q = p$, then $\Delta_{pp}^{(i)} = 0$.*
- *If q is nonfaulty through period i, then $|\Delta_{qp}^{(i)}| \leq S$ and*
$$|c_p^{(i)}(T' + \Delta_{qp}^{(i)}) - c_q^{(i)}(T')| < \epsilon$$
for some time T' in $S^{(i)}$.
- *If q is manifest-faulty in period i, then $\Delta_{qp}^{(i)} = 0$.*
- *If q is symmetric-faulty in period i, then*
$$|\Delta_{qp}^{(i)} - \Delta_{qr}^{(i)}| < \Delta$$
for all nonfaulty processors r.
- *If q is arbitrary-faulty in period i, no constraints are placed on the value $\Delta_{qp}^{(i)}$.*

2.3 Analysis of the Modified Algorithm

The main change in the analysis of the algorithm occurs in Lemma 5.

Lemma 5 (Version for Hybrid Fault Model). *If the clock synchronization conditions S1 and S2 hold for i, processors p and q are nonfaulty through period $i+1$, and $T \in S^{(i)}$, then*

- *If processor r is manifest-faulty in period i, then*
$$|c_p^{(i)}(T) + \bar{\Delta}_{rp}^{(i)} - [c_q^{(i)}(T) + \bar{\Delta}_{rq}^{(i)}]| < \delta$$
- *If processor r is symmetric-faulty in period i, then*
$$|c_p^{(i)}(T) + \bar{\Delta}_{rp}^{(i)} - [c_q^{(i)}(T) + \bar{\Delta}_{rq}^{(i)}]| < \delta + \Delta$$
- *If processor r is arbitrary-faulty in period i, then*
$$|c_p^{(i)}(T) + \bar{\Delta}_{rp}^{(i)} - [c_q^{(i)}(T) + \bar{\Delta}_{rq}^{(i)}]| < \delta + 2\Delta.$$

Proof: For manifest- and arbitrary-faulty r, we can write

$$\begin{aligned} &|c_p^{(i)}(T) + \bar{\Delta}_{rp}^{(i)} - [c_q^{(i)}(T) + \bar{\Delta}_{rq}^{(i)}]| \\ &= |c_p^{(i)}(T) - c_q^{(i)}(T) + \bar{\Delta}_{rp}^{(i)} - \bar{\Delta}_{rq}^{(i)}| \\ &\leq |c_p^{(i)}(T) - c_q^{(i)}(T)| + |\bar{\Delta}_{rp}^{(i)}| + |\bar{\Delta}_{rq}^{(i)}|. \end{aligned}$$

and the result follows on applying S1 to the first term on the right hand side, and observing that the Algorithm ensures that the remaining two terms are zero for manifest-faulty r, and no larger than Δ for arbitrary-faulty r. For symmetric-faulty r, we use the inequality

$$\begin{aligned} &|c_p^{(i)}(T) + \bar{\Delta}_{rp}^{(i)} - [c_q^{(i)}(T) + \bar{\Delta}_{rq}^{(i)}]| \\ &\leq |c_p^{(i)}(T) - c_q^{(i)}(T)| + |\bar{\Delta}_{rp}^{(i)} - \bar{\Delta}_{rq}^{(i)}|. \end{aligned}$$

S1 ensures that the first term on the right hand side is no greater than δ; by expanding the definitions of $\bar{\Delta}_{rp}^{(i)}$ and $\bar{\Delta}_{rq}^{(i)}$ (in terms of $\Delta_{rp}^{(i)}$ and $\Delta_{rq}^{(i)}$, respectively), and applying the constraint for symmetric-faulty processors from A2, we see that the second term is no larger than Δ, and the result follows. □

If there are a arbitrary, s symmetric, and m manifest faults, and we set $t = a+s+m$ (i.e., t is the total number of faults), then the maximum clock skew between two nonfaulty processors in any point in the period following resynchronization is bounded by

$$\frac{(n-t)(2[\epsilon + \rho S] + \rho\Delta)}{n} + \frac{a(\delta + 2\Delta)}{n} + \frac{s(\delta+\Delta)}{n} + \frac{m\delta}{n} + \rho(R + \Sigma).$$

To satisfy S1, this must be no greater than δ, so we obtain the following version of Constraint C6.

Constraint C6 (for Hybrid Fault Model):

$$\delta \geq 2(\epsilon + \rho[S + \frac{\Delta}{2}]) + \frac{(2a+s)\Delta}{n-t} + \frac{n\rho(R+\Sigma)}{n-t}.$$

It is easy to check that this can be satisfied simultaneously with C4 only if $n > 3a + 2s + m$.

2.4 Extensions to the Fault Model

There are a number of useful extensions that can be made to the hybrid fault model. As presently formulated, a manifest-faulty processor must appear so to all nonfaulty processors. Communications faults that cause just one or two processors to receive manifestly erroneous clock readings must, because of the asymmetry in observed behavior, be counted as arbitrary faults in one or more of the processors: thus a common and simple kind of fault seems to be rather expensive to tolerate. In our version of the Oral Messages algorithm for distributed consensus under the hybrid fault model [8], this does indeed seem to be the case (at least, in the presence of arbitrary and symmetric faults); for clock synchronization, however, these kinds of faults do not exact the same cost. I present two treatments: one widens the class of faults considered as symmetric, the other introduces link faults as a new fault mode.

2.4.1 Extending the Symmetric Fault Mode

Notice that it is possible to satisfy the symmetric-fault constraint $|\Delta_{q\,p}^{(i)} - \Delta_{q\,r}^{(i)}| < \Delta$ when one or both of $\Delta_{q\,p}^{(i)}$ and $\Delta_{q\,r}^{(i)}$ exceeds Δ (as might happen if processor q has lost synchronization). In this case, the ordinary algorithm will set one or both of $\bar{\Delta}_{q\,p}^{(i)}$ or $\bar{\Delta}_{q\,r}^{(i)}$ to zero. If the algorithm still works in this case (and we have just seen that it does), then it will also work if either of $\Delta_{q\,p}^{(i)}$ or $\Delta_{q\,r}^{(i)}$ had themselves been zero—as would happen if processor r was asymmetrically (or intermittently) manifest-faulty. We can add this case to the definition of the symmetric fault mode by changing the corresponding clause (i.e., the fourth bullet) of the hybrid version of Assumption A2 to read

- *If q is symmetric-faulty in period i, then either $\Delta_{q\,p}^{(i)} = 0$ or $\Delta_{q\,r}^{(i)} = 0$ or*
$$|\Delta_{q\,p}^{(i)} - \Delta_{q\,r}^{(i)}| < \Delta$$
for all nonfaulty processors r.

The only adjustment required to accommodate this change is the addition of a new, and straightforward case in the proof of Lemma 5.

This treatment reduces the cost of tolerating an asymmetrically manifest-faulty processor from that of the arbitrary to the symmetric fault mode. However, it has the disadvantage of treating the afflicted processor as faulty and therefore fails to discuss whether that processor can stay synchronized. This will be of interest if the physical source of the faulty behavior is in the communications links rather than in the processor itself. To provide a better treatment for faults in the communications links, it is necessary introduce a new fault mode explicitly for that case.

2.4.2 Link Faults

I will say that a *link* (q,p) is faulty if the value $\Delta_{q\,p}^{(i)}$ can appear as if q were a manifest-faulty processor (i.e., the value can be 0). The strictest fault model would stipulate that the value should *always* appear manifest-faulty. However, the cost of tolerating this fault mode will be in the asymmetries between $\Delta_{q\,p}^{(i)}$ and $\Delta_{q\,r}^{(i)}$, where (q,r) is a *non*faulty link, and the corresponding effects on the adjustments made by processors p and r. This asymmetry is not increased if we allow faulty links to be intermittent—that is to behave sometimes correctly and sometimes as a source of manifest faults. Thus, I introduce link faults to the hybrid version of Assumption A2 by amending the second bullet (q nonfaulty) to require that the link (q,p) is nonfaulty in period i, and adding the following the clause for the case where the link is faulty.

Assumption A2 (extra clause for link faults):

- *If q is nonfaulty through period i, and (q,p) is link-faulty at period i, then either $\Delta_{q\,p}^{(i)} = 0$ or*
$$|\Delta_{q\,p}^{(i)}| \le S$$
and
$$|c_p^{(i)}(T' + \Delta_{q\,p}^{(i)}) - c_q^{(i)}(T')| < \epsilon$$
for some time T' in $S^{(i)}$.

Analysis of these faults is undertaken in a modification to Lemma 4.

Lemma 4 (Version for Link Faults). *If the clock synchronization conditions S1 and S2 hold for i, processors p and q are nonfaulty through period $i{+}1$, processor r is nonfaulty through period i, one or both of (r,p) and (r,q) is link-faulty in period i, and $T \in S^{(i)}$, then*

$$|c_p^{(i)}(T) + \bar{\Delta}_{r\,p}^{(i)} - [c_q^{(i)}(T) + \bar{\Delta}_{r\,q}^{(i)}]| < \delta + \epsilon + \rho S + \frac{\rho}{2}\Delta.$$

Proof: (Sketch; note, the result is true even if r is one of p or q.) If neither (r,p) nor (r,q) displays faulty behavior, the regular Lemma 4 applies and the result follows, provided $\delta \ge \epsilon + \rho S + \frac{\rho}{2}\Delta$. This constraint is examined below.

If both (r,p) and (r,q) display faulty behavior, then the case of Lemma 5 for manifest-faulty r applies and the result follows immediately.

If exactly one of (r,p) and (r,q) displays faulty behavior, then using the arithmetic identity

$$x - y = (u - y) - (v - x) + (v - w) - (u - w)$$

we obtain

$$\begin{aligned}
&|c_p^{(i)}(T) + \Delta_{r\,p}^{(i)} - [c_q^{(i)}(T) + \Delta_{r\,q}^{(i)}]| \\
&\quad = \ |c_q^{(i)}(T + \Delta_{r\,q}^{(i)}) - [c_q^{(i)}(T) + \Delta_{r\,q}^{(i)}] \\
&\qquad - (c_p^{(i)}(T + \Delta_{r\,p}^{(i)}) - [c_p^{(i)}(T) + \Delta_{r\,p}^{(i)}]) \\
&\qquad + c_p^{(i)}(T + \Delta_{r\,p}^{(i)}) - c_r^{(i)}(T) \\
&\qquad - (c_q^{(i)}(T + \Delta_{r\,q}^{(i)}) - c_r^{(i)}(T))|.
\end{aligned}$$

Without loss of generality, suppose it is (r,q) that is faulty, so that $\Delta_{r\,q}^{(i)} = 0$. Making this substitution, rearranging terms, and canceling, we obtain

$$\begin{aligned}
&|c_p^{(i)}(T) + \Delta_{r\,p}^{(i)} - [c_q^{(i)}(T) + \Delta_{r\,q}^{(i)}]| \\
&\quad \le \ |c_r^{(i)}(T) - c_q^{(i)}(T)| \\
&\qquad + \ |c_p^{(i)}(T + \Delta_{r\,p}^{(i)}) - [c_p^{(i)}(T) + \Delta_{r\,p}^{(i)}]| \\
&\qquad + \ |c_p^{(i)}(T + \Delta_{r\,p}^{(i)}) - c_r^{(i)}(T)|.
\end{aligned}$$

By S1, the first term on the right hand side is no greater than δ. Using Lemmas numbered 1 and 2d from the formal verification of the classical ICA algorithm [15], the second term on the right hand side can be shown to be no greater than $\frac{\rho}{2}\Delta$, and Lemma 3 from that verification shows that the final term is no greater then $\epsilon+\rho S$. The result follows. □

For two nonfaulty processors p and q, let $l_{p\,q}$ be the number of nonfaulty processors r such that one or both of (r,p) and (r,q) is link-faulty (note that p or q can be such an r). Then the worst-case clock skew between two nonfaulty processors at any point in the period following resynchronization is bounded by

$$\frac{(n-a-s-m-\frac{l_{p\,q}}{2})(2[\epsilon+\rho S]+\rho\Delta)}{n} + \frac{a(\delta+2\Delta)}{n} + \frac{s(\delta+\Delta)}{n} + \frac{l_{p\,q}\delta+m\delta}{n} + \rho(R+\Sigma).$$

If we then let l be the maximum over all p and q of $l_{p\,q}$ (so that l is at most the number of nonfaulty processors having one or more faulty "outgoing" links), define $t = a+s+m+l$, and again assume $\delta \geq \epsilon+\rho S+\frac{\rho}{2}\Delta$, this becomes

$$\frac{(n-t+\frac{l}{2})(2[\epsilon+\rho S]+\rho\Delta)}{n} + \frac{a(\delta+2\Delta)}{n} + \frac{s(\delta+\Delta)}{n} + \frac{l\delta+m\delta}{n} + \rho(R+\Sigma),$$

which yields the following version of the Constraint C6.

Constraint C6 (for Hybrid Fault Model with Link Faults):

$$\delta \geq \frac{2(n-t+\frac{l}{2})(\epsilon+\rho[S+\frac{\Delta}{2}])+(2a+s)\Delta+n\rho(R+\Sigma)}{n-t}.$$

Taking this with Constraint C4, we can deduce that a necessary (though not sufficient) condition for the existence of a solution is

$$n > 3a+2s+m+l.$$

It remains to discharge the constraint $\delta > \epsilon+\rho S+\frac{\rho}{2}\Delta$ used above. This follows from the version of C6 just derived, because $n-t \geq 2$ (otherwise we cannot discuss synchronization).

3 Formal Verification

The real difficulty in proving correctness of ICA and its variants is not in the arithmetic manipulations performed in the proofs of Lemmas such as 4 and 5, but in making the whole argument hang together. This argument is a delicate web of interconnected constraints, assumptions, and lemmas, whose hypotheses must be stated very carefully. For example, Lemma 4 is required in the proof of S1, its own proof requires Assumption A2, A2 requires S1, and all are bound together in an induction. Furthermore, it is necessary to be careful about the exact durations of the periods in which a processor is assumed nonfaulty. The flaws in Lamport and Melliar-Smith's proof of correctness of ICA [5] were mostly in these details. The mechanically-checked formal verification of ICA that von Henke and I constructed in 1988 led us to detect and correct these flaws [15, 16]. Our corrected proofs change some of the assumptions and the statements of the lemmas, and also eliminate the approximations used by Lamport and Melliar-Smith, yielding correct proofs that provide simple and exact bounds and that are easier to follow than the originals.

The improved argument for correctness, and the improved understanding of that argument that derives from having undertaken formal verification, made it quite easy to see how to adjust matters to accommodate a hybrid fault model. Furthermore, although construction of the formal verification for the original algorithm was a significant effort (at the time, it was one of the hardest mechanically-checked verifications that had been performed[2]), subsequent tinkering has proved relatively inexpensive. Moreover, it is reliable and safe—which modifications to an informal argument seldom are: a mechanized theorem prover brings the same implacable skepticism to its second and third (or hundredth) encounter with a given problem as to its first.

Our formal verification of ICA has been adjusted several times, as new requirements or opportunities were discovered. For example, the original verification incorporated an assumption from Lamport and Melliar-Smith that the initial clock corrections $C_p^{(0)}$ are all zero. Later, when Liu and I were designing a circuit to read clock differences [9], we realized that this was a very inconvenient constraint and wondered if it could be eliminated. We explored this possibility by simply eliminating the constraint from the formal specification and rerunning all the proofs (which takes about 20 minutes on a SPARC2). We found that the proofs of a few internal lemmas failed, but that they could be adjusted to use slightly different arguments without affecting the rest of the verification.

Another adjustment to the verification derived from the discovery, made by Palumbo and Graham in mea-

[2] It is still a formidable challenge to most theorem proving systems because of the extensive arithmetic reasoning that is required. Only systems with integrated decision procedures for real and integer arithmetic stand any chance of completing the proof at reasonable cost. (The full proof has over 200 mechanically-checked lemmas.) It has been repeated just once: by Young [22], using the Boyer-Moore prover.

surements of a hardware implementation of the interactive convergence algorithm, that the observed worst-case skew is significantly better than that predicted by the standard Constraint C6 [13]. They showed that observation and theory could be brought into much closer agreement by using the fact that there is no error when a processor reads its own clock. The improved bound is easily incorporated into the formal verification by noting that the constraint in Lemma 4 is halved when processor r is one of p or q. This allows the factor 2 in the first term of C6 to reduced to $2\frac{n-t-1}{n-t}$. Because the parameters to the algorithm are interrelated, this adjustment also permits a reduction in the bound on Δ, which is significant for the second term in C6. Overall, these adjustments yield improvements of 10% to 25% in numerical estimates of the worst-case skew.

Extending the formal verification of ICA to accommodate Palumbo and Graham's improved bound required development of a body of lemmas concerning finite summations. Previously, it had only been necessary to split the summation over clock differences into two—according to whether processor r is faulty or nonfaulty—and an ad-hoc treatment was adequate. For Palumbo and Graham's bound, it was necessary to split the summation further according to whether r is the same as one of p or q, and it seemed better to develop a more general theory of summations. Availability of this general theory made it feasible to contemplate hybrid faults (where the summation needs to be further subdivided according to fault mode).

For the hybrid algorithm, it was the work of a couple of hours to adjust the formal specification and develop a mechanically-checked proof for the hybrid version of Lemma 5 (an excerpt is reproduced in the appendix). However, a whole day was spent without success trying to carry through the full proof for the hybrid case (without link faults). Eventually, the source of the problem was found in an unexpected location: in the statement of Lemma 4 (which is unchanged from the classical case when link faults are not considered). Following Lamport and Melliar-Smith, the hypotheses to our version of this lemma require that processors p, q, and r are nonfaulty through period $i+1$. Because the lemma is concerned with the skew between processors p and q in the $i+1$st period, these processors must indeed be nonfaulty in this period. However, the role of processor r is to provide its clock differences to processors p and q in the resynchronization that occurs at the end of period i. Thus, it is only necessary that processor r should be nonfaulty through period i. With this adjustment, the hypotheses of the standard version of Lemma 4 coincide with those of the hybrid version of Lemma 5 and the full proof can be completed. The overly strong hypothesis in the original Lemma 4 concerning processor r did not affect verification of the classical algorithm because that makes no assumptions about the behavior of faulty processors (thus the condition of processor r is not mentioned in the hypotheses to the original Lemma 5). The new algorithm and its verification have to consider the mode of failure of processor r at period i in Lemma 5, and so Lemma 4 has to be adjusted accordingly.

4 Conclusions

The work presented here widens the fault model for clock synchronization to include three realistic fault modes—manifest, symmetric, and link faults—in addition to the arbitrary or Byzantine case. A simple adjustment to the Interactive Convergence algorithm, and slightly more complicated adjustments to its analysis, yield a formally verified algorithm that performs well under each of these fault modes and their combinations. Among synchronization algorithms suitable for the architectures used in digital flight control, this algorithm seems to provide the most robust fault tolerance for a given level of redundancy.

Although the main change needed to accommodate the hybrid fault model (without link faults) is in Lemma 5, formal verification revealed the unexpected need to adjust the statement of Lemma 4 also. I consider it unlikely that this would have been discovered in an ordinary journal-style proof. As it happens, this adjustment has no further repercussions, but that is mere good fortune. This experience reinforces the conviction I have acquired from undertaking many other verifications: the intricate correctness arguments for fault-tolerant algorithms are often flawed (and sometimes the algorithms have bugs); only mechanically-checked formal verification seems able to provide adequate assurance for these safety-critical algorithms. Small changes, even those with a convincing informal proof of correctness, should be viewed with suspicion and the full mechanically-checked verification should be repeated following any change. (Recall that the statement of Lemma 4 had been satisfactory in previous formal verifications; it only became inadequate when hybrid faults were considered.)

Fortunately, a formal specification and verification is a reusable intellectual resource, and it can be relatively inexpensive—as well as reliable and safe—to adjust a mechanically-checked verification to accommodate changed assumptions or requirements. I believe this is an inadequately recognized benefit of formal verification. Other benefits that are similarly underestimated by those who see mechanically-checked formal analysis only in terms of "proof of correctness" include: debugging (i.e., discovery of *in*correctness), complete enumeration of assumptions, sharpened statements of assumptions and lemmas, streamlined arguments, and an enhanced understanding that can lead to further improvements.

For future investigations, I suggest extending other clock synchronization algorithms in a similar manner to that described here. Shankar's mechanically-checked formal verification [19] of Schneider's general treatment [18] would be a good starting place for formal investigation of such extensions (particularly with Miner's formally verified extension to transient-recovery for the Welch-Lynch instantiation [11, 21]).

Another interesting topic for future investigation is the apparent divergence in fault-tolerance between algorithms for consensus and those for clock synchronization when hybrid fault models are considered. As we have seen, the hybrid interactive convergence algorithm for clock synchronization is able to withstand a arbitrary, s symmetric, and m manifest faults simultaneously, provided n, the number of processors satisfies $n > 3a + 2s + m$, whereas the r-round version of the hybrid Oral Messages algorithm for consensus requires $n > 2a + 2s + m + r$ and $a \leq r$ [8]. When only arbitrary faults are present, we have $a = r$ and $n > 3a$, so both algorithms exhibit optimal fault tolerance. But when only manifest or symmetric faults are present, the hybrid Oral Messages algorithm appears suboptimal (in that additional rounds decrease the number of faults that can be tolerated). In fact, separate analysis shows that the number of faults tolerated by hybrid Oral Messages is optimal when only manifest faults are present ($n - 1$ can be tolerated, independently of the number of rounds), but that the suboptimality is real when only symmetric faults are present (if $r > 0$). Furthermore, the hybrid Oral Messages algorithm does not appear able to tolerate link faults, whereas hybrid interactive convergence does so inexpensively. It seems that the approximate nature of clock synchronization is helpful in these cases, and an interesting direction for future work is to examine whether the approximate agreement version of distributed consensus [1] is better suited to the hybrid fault model than exact agreement.

Acknowledgement Ricky Butler of NASA Langley Research Center modified the original formal specification and verification of the Interactive Convergence algorithm to make better use of the capabilities of recent versions of EHDM.

References

[1] Danny Dolev, Nancy A. Lynch, Shlomit S. Pinter, Eugene W. Stark, and William E. Weihl. Reaching approximate agreement in the presence of faults. *Journal of the ACM*, 33(3):499–516, July 1986.

[2] Juan A. Garay and Kenneth J. Perry. A continuum of failure models for distributed computing. In A. Segall and S. Zaks, editors, *Distributed Algorithms (6th International Workshop WDAG '92)* (Haifa, Israel), pages 153–165, November 1992. Volume 647 of *Lecture Notes in Computer Science*, Springer-Verlag.

[3] A. H. Infis and W. R. Moore. Economic approach to fault-tolerant synchronization. *IEE Proceedings, Part E: Computers and Digital Techniques*, 135(2):82–86, March 1988.

[4] R. M. Kieckhafer, C. J. Walter, A. M. Finn, and P. M. Thambidurai. The MAFT architecture for distributed fault tolerance. *IEEE Transactions on Computers*, 37(4):398–405, April 1988.

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

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

[7] 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* (Elounda, Greece), pages 292–304, June/July 1993. Volume 697 of *Lecture Notes in Computer Science*, Springer-Verlag.

[8] Patrick Lincoln and John Rushby. A formally verified algorithm for interactive consistency under a hybrid fault model. In *Fault Tolerant Computing Symposium 23*, pages 402–411, June 1993. IEEE Computer Society.

[9] Erwin Liu and John Rushby. A formally verified module to support Byzantine fault-tolerant clock synchronization. Project report 8200-130, Computer Science Laboratory, SRI International, Menlo Park, CA, December 1993.

[10] 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.

[11] Paul S. Miner. Verification of fault-tolerant clock synchronization systems. NASA Technical Paper 3349, NASA Langley Research Center, Hampton, VA, November 1993.

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

[13] Daniel L. Palumbo and R. Lynn Graham. Experimental validation of clock synchronization algorithms. NASA Technical Paper 2857, NASA Langley Research Center, Hampton, VA, July 1992.

[14] Kenneth J. Perry and Sam Toueg. Distributed agreement in the presence of processor and communication faults. *IEEE Transactions on Software Engineering*, SE-12(3):477–481, March 1986.

[15] 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.

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

[17] John Rushby, Friedrich von Henke, and Sam Owre. An introduction to formal specification and verification using EHDM. Technical Report SRI-CSL-91-2, Computer Science Laboratory, SRI International, Menlo Park, CA, February 1991.

[18] Fred B. Schneider. Understanding protocols for Byzantine clock synchronization. Technical Report 87-859, Department of Computer Science, Cornell University, Ithaca, NY, August 1987.

[19] 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* (Nijmegen, The Netherlands), pages 217–236, January 1992. Volume 571 of *Lecture Notes in Computer Science*, Springer-Verlag.

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

[21] J. Lundelius Welch and N. Lynch. A new fault-tolerant algorithm for clock synchronization. *Information and Computation*, 77(1):1–36, April 1988.

[22] William D. Young. Verifying the Interactive Convergence clock-synchronization algorithm using the Boyer-Moore prover. NASA Contractor Report 189649, NASA Langley Research Center, Hampton, VA, April 1992. (Work performed by Computational Logic Incorporated).

Appendix: A Fragment of the Formal Specification and Verification

As an illustrative fragment of the text submitted to the EHDM formal verification system, the statement and proof of the manifest-faulty case of the hybrid version of Lemma 5 is reproduced below. The specification has been prettyprinted by EHDM. The text of the full verification is 110 pages long.

The list following the FROM keyword in the PROVE declaration enumerates the lemmas, axioms, and definitions to be used in the proof; the material in braces indicates substitutions to be made for quantified variables. Given this information, EHDM generates a ground formula by Skolemization and substitution for free variables, and then applies a decision procedure for a combination of theories including real and integer arithmetic and propositional calculus. The whole process takes only a couple of seconds.

lemma5_hybrid : **MODULE**

USING *lemma3_hybrid, lemma4, lemma5,*
  algorithm_hybrid, algorithm, clockprops
THEORY
  p, q, r : **VAR** *proc*
  T : **VAR** *clocktime*
  i, j : **VAR** *period*
  lemma5_manifest : **LEMMA**

$$
\begin{aligned}
& S_1C(p, q, i) \\
& \quad \wedge\ \mathrm{S2}(p, i) \\
& \quad\quad \wedge\ \mathrm{S2}(q, i) \\
& \quad\quad\quad \wedge\ \mathit{nonfaulty}(p, i+1) \\
& \quad\quad\quad\quad \wedge\ \mathit{nonfaulty}(q, i+1) \\
& \quad\quad\quad\quad\quad \wedge\ \mathit{fault\_type}(r, i) = \mathit{manifest}\ \wedge\ T \in S^{(i)} \\
& \quad \supset\ |c_p^{(i)}(T) + \bar{\Delta}_{r\,p}^{(i)} - (c_q^{(i)}(T) + \bar{\Delta}_{r\,q}^{(i)})|\ \le\ \delta
\end{aligned}
$$

  lemma5_manifest_proof : **PROVE** *lemma5_manifest*
   FROM
    S1C,
    inRS,
    A2_manifest $\{q \leftarrow r\}$,
    A2_manifest $\{p \leftarrow q, q \leftarrow r\}$,
    $\bar{\Delta}_{r\,p}^{(i)}, \bar{\Delta}_{r\,q}^{(i)}$,
    nonfx, *nonfx* $\{p \leftarrow q\}$,
    abs_ax0
  ...
END *lemma5_hybrid*