A PVS Specification of OMH-FTP

$omhftp[T:\ \text{TYPE},\ error:\ T,\ num:\ above[2],\ R,\ UnR:\ [T \to T]]:\ \text{THEORY}$
BEGIN
ASSUMING $R_ax:\ \text{ASSUMPTION}\ (\forall\ (t:\ T):\ R(t) \neq error)$
$UnR_ax:\ \text{ASSUMPTION}\ (\forall\ (t:\ T):\ UnR(R(t)) = t)$
ENDASSUMING
$t:\ \text{VAR}\ T$
$n:\ \text{VAR}\ below[num]$
$processor:\ \text{TYPE} = below[num]$
$interstage:\ \text{TYPE} = below[num]$
$G, p, q:\ \text{VAR}\ processor$
$iv,\ iv2:\ \text{VAR}\ [interstage \to T]$
$i:\ \text{VAR}\ interstage$
IMPORTING $finite_cardinality[interstage,\ num, (\lambda\ (x:\ below[num]):\ x)],$
$finite_cardinality[processor,\ num, (\lambda\ (x:\ below[num]):\ x)],$
$filters[below[num]],$
$card_set[interstage,\ num, (\lambda\ (x:\ below[num]):\ x)],$
$card_set[processor,\ num, (\lambda\ (x:\ below[num]):\ x)]$
$proc(i:\ interstage):\ processor = i$
$statuses:\ \text{TYPE} = \{good,\ arbitrary,\ manifest,\ symmetric\}$
$pstatus:\ [processor \to statuses]$
$istatus:\ [interstage \to statuses]$
$gp(p):\ bool = good(pstatus(p))$
$gi(i):\ bool = good(istatus(i))$
$ap(p):\ bool = arbitrary(pstatus(p))$
$ai(i):\ bool = arbitrary(istatus(i))$
$mp(p):\ bool = manifest(pstatus(p))$
$mi(i):\ bool = manifest(istatus(i))$
$sp(p):\ bool = symmetric(pstatus(p))$
$si(i):\ bool = symmetric(istatus(i))$
$gpair(i):\ bool = gi(i) \wedge gp(proc(i))$
$bpair(i):\ bool = ai(i) \vee si(i) \vee ap(proc(i)) \vee sp(proc(i))$

$sendpti:\ [T,\ processor,\ interstage \to T]$
$senditp:\ [T,\ interstage,\ processor \to T]$
$sendgtp:\ [T,\ processor,\ processor \to T]$

$sendgtp_ax1:\ \text{AXIOM}\ gp(p) \supset sendgtp(t, p, q) = t$
$sendpti_ax1:\ \text{AXIOM}\ gp(p) \supset sendpti(t, p, i) = t$
$senditp_ax1:\ \text{AXIOM}\ gi(i) \supset senditp(t, i, p) = t$
$sendgtp_ax2:\ \text{AXIOM}\ mp(p) \supset sendgtp(t, p, q) = error$
$sendpti_ax2:\ \text{AXIOM}\ mp(p) \supset sendpti(t, p, i) = error$
$senditp_ax2:\ \text{AXIOM}\ mi(i) \supset senditp(t, i, p) = error$
$sendgtp_ax3:\ \text{AXIOM}\ sp(G) \supset sendgtp(t, G, p) = sendgtp(t, G, q)$
$senditp_ax3:\ \text{AXIOM}\ si(i) \supset senditp(t, i, p) = senditp(t, i, q)$
$sendgtp_ax4:\ \text{LEMMA}\ \neg\ ap(G) \supset sendgtp(t, G, p) = sendgtp(t, G, q)$
$senditp_ax4:\ \text{LEMMA}\ \neg\ ai(i) \supset senditp(t, i, p) = senditp(t, i, q)$

$H_Majority(iv):\ T$

$Majority_ax1:\ \text{AXIOM}$
$|gpair| > |bpair| \wedge (\forall\ i:\ gpair(i) \supset iv(i) = t) \wedge t \neq error$
$\wedge\ (\forall\ i:\ mi(i) \supset iv(i) = error) \wedge (\forall\ i:\ gi(i) \wedge mp(proc(i)) \supset iv(i) = error)$
$\supset Majority(iv) = t$

$Majority_ax2:\ \text{AXIOM}(\forall\ i:\ iv(i) = iv2(i)) \supset H_Majority(iv) = H_Majority(iv2)$

$OMH_FTP(G, t)(p):\ T = UnR(H_Majority(\lambda\ i: senditp(sendpti(R(sendgtp(t, G,\ proc(i))), proc(i), i), i, p)))$

$Validity_Final:\ \text{THEOREM}$
$gp(p) \wedge \neg\ ap(G) \wedge num > 2 \times (|ap| + |ai| + |sp| + |si|) + |mi| + |mp|$
$\supset OMH_FTP(G, t)(p) = sendgtp(t, G, p)$

$Agreement_Final:\ \text{THEOREM}$
$gp(p) \wedge gp(q) \wedge |ap| + |ai| \leq 1 \wedge num > 2 \times (|ap| + |ai| + |sp| + |si|) + |mi| + |mp|$
$\supset OMH_FTP(G, t)(p) = OMH_FTP(G, t)(q)$

END $omhftp$

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

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

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

[18] 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, Toulouse, France, June 1993. IEEE Computer Society.

[19] 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.

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

[21] 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.

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

[23] 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.

[24] John Rushby. A fault-masking and transient-recovery model for digital flight-control systems. In Jan Vytopil, editor, *Formal Techniques in Real-Time and Fault-Tolerant Systems*, Kluwer International Series in Engineering and Computer Science, chapter 5, pages 109–136. Kluwer, Boston, Dordecht, London, 1993. An earlier version appeared in [?, pp. 237–257].

[25] John Rushby. A formally verified algorithm for clock synchronization under a hybrid fault model. In *Thirteenth ACM Symposium on Principles of Distributed Computing*, Los Angeles, CA, August 1994. Association for Computing Machinery. (To appear).

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

[27] 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.

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

[29] 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.

[30] 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.

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

rather tedious, and we plan to investigate use of state-exploration techniques for this purpose.

This demonstration adds to the collection of formally specified and verified system components in NASA Langley's "reliable computing platform" for ultra-critical applications [4]. The most recent additions to this set, including Hybrid Interactive Consistency [16], Hybrid Clock Synchronization [25], and the algorithm examined here, demonstrate the practical feasibility of formal verification for intricate problems where human reasoning and peer review has been found somewhat unreliable.

These examples also demonstrate the value of formal specifications and verifications in supporting reuse and enhancements of existing designs and algorithms: the primary benefit we have found in formal methods is not in the verification of an individual algorithm, but in the opportunity such a verification creates for the reliable exploration of alternative assumptions, requirements and designs.

References

[1] A. Avižienis, H. Kopetz, and J. C. Laprie, editors. *The Evolution of Fault-Tolerant Computing*, volume 1 of *Dependable Computing and Fault-Tolerant Systems*. Springer-Verlag, Wien, Austria, 1987.

[2] 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.

[3] Robert S. Boyer and J Strother Moore. MJRTY—a fast majority vote algorithm. In Robert S. Boyer, editor, *Automated Reasoning: Essays in Honor of Woody Bledsoe*, volume 1 of *Automated Reasoning Series*, pages 105–117. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1991.

[4] Ricky W. Butler. NASA Langley's research program in formal methods. In *COMPASS '91 (Proceedings of the Sixth Annual Conference on Computer Assurance)*, pages 157–162, Gaithersburg, MD, June 1991. IEEE Washington Section.

[5] Ben L. Di Vito and Ricky W. Butler. Formal techniques for synchronized fault-tolerant systems. In C. E. Landwehr, B. Randell, and L. Simoncini, editors, *Dependable Computing for Critical Applications—3*, pages 163–188, Mondello, Sicily, Italy, September 1992. Volume 8 of *Dependable Computing and Fault-Tolerant Systems*, Springer-Verlag, Wien, Austria.

[6] Ben L. Di Vito, Ricky W. Butler, and James L. Caldwell. High level design proof of a reliable computing platform. In J. F. Meyer and R. D. Schlichting, editors, *Dependable Computing for Critical Applications—2*, pages 279–306, Tucson, AZ, February 1991. Volume 6 of *Dependable Computing and Fault-Tolerant Systems*, Springer-Verlag, Wien, Austria.

[7] 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)*, pages 153–165, Haifa, Israel, November 1992. Volume 647 of *Lecture Notes in Computer Science*, Springer-Verlag.

[8] J. Goldberg. A history of research in fault-tolerant computing at SRI International. In Avižienis et al. [1], pages 101–119.

[9] Richard E. Harper and Jaynarayan H. Lala. Fault-tolerant parallel processor. *AIAA Journal of Guidance, Control, and Dynamics*, 14(3):554–563, May–June 1991.

[10] Albert L. Hopkins, Jr., Jaynarayan H. Lala, and T. Basil Smith III. The evolution of fault tolerant computing at the Charles Stark Draper Laboratory, 1955–85. In Avižienis et al. [1], pages 121–140.

[11] 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.

[12] Jaynarayan H. Lala. A Byzantine resilient fault tolerant computer for nuclear power application. In *Fault Tolerant Computing Symposium 16*, pages 338–343, Vienna, Austria, July 1986. IEEE Computer Society.

[13] Jaynarayan H. Lala, Richard E. Harper, and Linda S. Alger. A design approach for ultrareliable real-time systems. *IEEE Computer*, 24(5):12–22, May 1991.

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

or whole proofs, when working on variations of problems we have studied previously.

The proof of the Validity property was constructed with more difficulty, mostly because one part of it depends on reasoning about the hybrid fault model, and another part on the asymmetric FTP architecture. Consequently, proof fragments were copied from our earlier proofs of Validity for the n-plex OM-FTP, and from older proofs of OMH [17]. The main difficulty then lay in choosing where to put the correct fragments, and we made some poor choices initially, which led to our reconstructing large parts of the proof from scratch. We should remark that the interactive proof checker of PVS is quite a pleasure to use, but it is certainly more time consuming to develop a whole proof interactively than to simply copy a proof transcript.

After developing this formal specification and verification, we explored a few alternate statements of lemmas and axioms, but did not attempt the generalization developed for n-plex OM-FTP to include processors without interstages. We chose not to do this because with the hybrid fault model one can simply consider any missing interstage as manifest-faulty. We intend to continue exploring minor changes in formulation as well as more fundamental changes to the algorithm, architecture, and fault model using the same copy-and-edit paradigm for the creation of formal specifications and formal proofs.

3.4 Discussion

In our formal investigation of the FTP architecture, we have found previous claims of its advantages to be verifiably correct, but too modest. That is, the statements are true, but usefully stronger statements are true as well. For example, we found that the architecture performs adequately even when some processors have no associated interstage. Furthermore, algorithms such as OMH-FTP can use the architecture to resist larger numbers of faults than previously claimed for OM-FTP.

We believe that our insights and more exact analysis are the direct result of the process of formal specification and verification. Without formal specification and verification, it is extremely difficult to comprehend the consequences of simultaneous variations in an architecture, algorithm, and fault model. It is human nature to gloss over details when performing a re-analysis under some subtly different set of assumptions, and thus it is human nature to make mistakes in this process. It requires superhuman attention to the smallest detail of every case to perform these tasks correctly.

Fortunately, computers are exceedingly proficient at recording and checking such details. In a properly engineered interactive verification system such as PVS, a user is able to replay earlier proofs under different sets of assumptions and can quickly understand where the argument goes awry. The ability to deal quickly with routine parts of a verification, so that human insight can be concentrated on the more challenging aspects, is one of the requirements for moving formal methods beyond toy problems to the practical analysis of real systems design challenges.

Combining the (irreplaceable) high-level understanding of a human with the tirelessly meticulous low-level checking of a computer yields an extremely effective team, in our experience. The products of these human-computer collaborations include not only machine-checked proofs, but more importantly a deeper understanding of the problem.

4 Conclusion

We have described a high-level system architecture that combines the reduced hardware cost and complexity of Draper Laboratory's FTP architecture with the improved reliability (due to the tolerance of larger numbers of simple faults) of algorithms developed for hybrid fault models. We have formally specified and verified a traditional triplex FTP architecture under a standard Byzantine fault model, and extended this specification and verification using PVS's proof development tools to the n-plex case under a hybrid fault model. This system architecture and algorithm provide the best overall tradeoff we are aware of in Byzantine-fault-tolerant architectures.

In the future, we hope to examine related fault models, including link faults: a fault mode where receivers may receive either the correct value or a manifestly bad value. This class of faults has been successfully analyzed with respect to clock synchronization [25]. Adding authentication to OMH(m) does increase resilience against this type of fault, but it is not easy to characterize the fault tolerance achieved. In addition, our OMH-FTP algorithm does not make use of all the opportunities for single-round information exchange that are present in FTP (e.g., exchanges among the processors, or incorporating messages received directly from the transmitter in the majority vote). Exploring whether these opportunities can be exploited to provide greater fault-tolerance could be

to *H_Majority* denotes the *set* of values obtained as i ranges over all interstages. This concise formulation is possible because PVS is a higher-order logic.

Specification of the *H_Majority* function is another key element reused from earlier efforts. Rather than specifying an algorithm for performing this operation, we simply axiomatize the two properties of the function required for this application (see *Majority_ax1* and *Majority_ax2* in the Appendix). The two axioms are stated in terms of the fault-status of system components, a choice that simplifies their application here, but slightly complicates the provision of a model. We have separately verified that an implementation based on the efficient Boyer-Moore MJRTY algorithm [3] satisfies these axioms [17].

We now turn our attention to the main properties we would like to verify: Validity and Agreement. The following specification corresponds to the property Validity stated informally earlier.

Validity_Final : THEOREM
$gp(p) \wedge \neg\, ap(G)$
$\wedge\ num > 2 \times (|ap| + |ai| + |sp| + |si|) + |mi| + |mp|$
$\supset\ OMH_FTP(G,t)(p) = sendgtp(t,G,p)$

Notations such as $|sp|$ represent the number of symmetric-faulty processors. The constant *num* is the number of processor-interstage pairs.

In English, one might read this specification as follows. If a processor p is nonfaulty, and transmitter G is not arbitrary-faulty, and there are more processor-interstage pairs than twice the number of arbitrary- and symmetric-faulty components plus the number of manifest-faulty components, then the algorithm achieves the same result as a simple message send from G to p. In the case that G is nonfaulty, then this is the correct value (t); if G is symmetric-faulty, this value is unknown, but is the same for all receivers.

The Agreement property is formalized as follows.

Agreement_Final : THEOREM
$gp(p) \wedge gp(q) \wedge |ap| + |ai| \leq 1$
$\wedge\ num > 2 \times (|ap| + |ai| + |sp| + |si|) + |mi| + |mp|$
$\supset\ OMH_FTP(G,t)(p) = OMH_FTP(G,t)(q)$

In English, one might read this as follows. If p and q are nonfaulty processors, and there is at most one arbitrary-faulty component in the system, and there are more processor-interstage pairs than twice the number of arbitrary- and symmetric-faulty components plus the number of manifest-faulty components, then after running the algorithm both p and q end up with the same value.

3.3 Formal Verification of OMH-FTP

Given the specification sketched above, which was quickly assembled from several previous sources, we set about formally verifying it. In some ways, the most interesting aspect of this project was the reuse of the formal proofs.

Using the PVS proof-editing facility, proof scripts from existing verifications of OM-FTP and OMH were combined as rough drafts for proofs of properties of OMH-FTP. This happened to be most successful for the Agreement property, but reuse of existing proof fragments was also instrumental in the rapid construction of a proof for the Validity property.

As an example, in developing the proof of the Agreement property, we began by copying verbatim an earlier proof description of the same property for the n-plex OM-FTP algorithm. We ran that proof, and encountered an immediate difficulty: the OM-FTP proof uses the proof command `(CASE "gp(G!1)")` to do a case split on whether the transmitter is nonfaulty. In case the transmitter is not nonfaulty, then in the original proof it must be arbitrary-faulty (because of the "Byzantine-only" fault model of the previous verification), and since there is at most one faulty component, no interstages can be faulty. This reasoning fails in the hybrid fault model, so we altered the command to split explicitly on whether the transmitter is arbitrary-faulty or not with the command `(CASE "not ap(G!1)")`. For the case the transmitter is arbitrary-faulty, the reused proof failed again for a similar reason: there is a deeper case split on whether a particular interstage is nonfaulty or not. The same repair and one other minor change was required to complete this branch of the proof. For the case the transmitter is not arbitrary-faulty, the original proof assumed the transmitter was nonfaulty, and thus sent consistent values. In the new proof we appeal instead to a lemma, proved separately, which states that non-arbitrary-faulty components always send consistent (although perhaps incorrect) values.

This proof-development process involved running the rough-drafts of proof scripts several times, making small changes to the script each time. On failure of a proof branch, PVS puts the user in the proof-context of the failed branch; the user can interactively construct the rest proof from there, or can go back to the proof-editor to make more global changes to the proof script. Of course, we could have built the proof of OMH-FTP from scratch, using our intuition from the informal proof to give direction, but we have found it much more efficient to re-use proof fragments

$$OMH_FTP(\mathrm{G,t})(\mathrm{p}) : \mathrm{T} = UnR(H_Majority(\lambda\ \mathrm{i} : senditp(sendpti(R(sendgtp(\mathrm{t}, \mathrm{G}, proc(\mathrm{i}))), proc(\mathrm{i}), \mathrm{i}), \mathrm{i}, \mathrm{p})))$$

Figure 2: PVS Specification of the Algorithm OMH-FTP

$(m'_p - 1))$ values equal to RE, and at least $m'_i + m'_p$ equal to E.

If the transmitter has a faulty interstage, then the transmitter is not counted among the m'_p, but its interstage is among those counted in a, s, or m_i. A nonfaulty processor therefore receives at least $n - 1 - ((a + s + m'_i - 1) + m'_p)$ values equal to RE, and at least $m'_i + m'_p$ equal to E.

In each case, the bound stated in the Lemma is sufficient to ensure that the value RE is returned in the *H-majority* vote in step (4). □

Theorem 1 *For any a, s, m, algorithm OMH-FTP satisfies the Agreement property if $n > 2(a + s) + m$ and $a \leq 1$.*

Proof: We first consider the case in which the transmitter is not arbitrary-faulty. Then Validity is ensured by Lemma 1, and we have agreement on the value actually sent or on E, depending whether or not the transmitter is manifest-faulty.

Now consider the case where the transmitter is arbitrary-faulty. There is at most one arbitrary-faulty component, so none of the receivers or interstages can be arbitrary-faulty. Thus, whatever each interstage does, whether it is faulty or not, it sends the same values to every processor. Each processor therefore receives the same set of values from the interstages, and will therefore obtain the same *H-majority* value in step (4) (remember, this value is functionally determined), thereby ensuring Agreement. □

3.2 Formal Specification of OMH-FTP

The formal specification of OMH-FTP was derived from those developed previously for OM-FTP and OMH(m). In this section we outline the specification and discuss some interesting aspects; the full specification appears in Appendix A.

As noted earlier, the key to all these specifications is the use of functions to model distributed behaviors, with the fault model encoded in a *send* function that models message transmission. Similar approaches have been used in several recent published formal verifications, but it has been noted that in some cases the use of functions to model message transmission actually introduces an extra (hidden) assumption of consistency on the behavior of Byzantine components in the later rounds of message passing (see [23] for more discussion of this point). In the present case the use of these functions introduces no such hidden assumption, since we only analyze the single-round case, and use separate *send* functions for each class of message transfers.

In particular, the specification of the algorithm OMH-FTP appears as the simple nonrecursive (but higher-order) function shown in Figure 2, where *sendgtp*, *sendpti*, and *senditp* model the transmission of messages from the transmitter (called the General in the formal specification) to the other processors (receivers), from processors to their interstages, and from interstages to the processors, respectively. (Both processors and interstages are represented by the integers $0, \ldots, num - 1$; this range is specified in PVS as *below*[*num*].)

The fault model is specified in eight axioms about these functions—such as the following, which captures the behavior of a symmetric-faulty transmitter.

$$sendgtp_ax3 : \text{AXIOM}\quad sp(G) \supset sendgtp(t, G, p) = sendgtp(t, G, q)$$

Here, *sp* is a predicate that is true if its argument is a symmetric-faulty processor. We also use *gp*, *ap*, and *mp* for nonfaulty ("good"), arbitrary-faulty ("Byzantine"), and manifest-faulty ("crashed") processors, respectively, and *si*, *gi*, *ai*, and *mi* for the corresponding classes of interstages.

The other functions appearing in the specification of *OMH_FTP* are *R*, *UnR*, *H_Majority*: these model the translation of E into RE and vice-versa, and the calculation of the *H-majority* (majority with E values excluded), respectively.

The function *OMH_FTP* is read most easily from inside to out: the transmitter G sends a value t to the processor associated with interstage i. This processor, *proc(i)*, then records the R of the value received and sends the result to its associated interstage i. Interstage i then sends the value received to all processors, which take the *H-majority* of all values received from interstages, and computes the *UnR* of the result. Note that the λ-abstraction appearing as the argument

3.1.4 The Algorithm

We use the Byzantine Generals formulation, in which there is a distinguished transmitter. In addition to the special value E that denotes the value received from a manifestly-faulty receiver or transmitter, we also introduce a special value RE that is used for "Reported Errors."

OMH-FTP

1. The transmitter sends its value to every receiver. If the transmitter has an interstage, it also sends its value to that interstage.
2. Each receiver with an interstage sends the value received to its interstage, *except* that if E was received, RE is sent to the interstage.
3. Each interstage relays the value it receives (if any) to every processor.
4. For each processor q, let $v_q(i)$ be the value received from interstage i. Processor q uses the value *H-majority*$(v_q(1), \ldots, v_q(n))$, which denotes the absolute majority in the set $\{v_q(i) | 1 \leq i \leq n, v_q(i) \neq E\}$ if it exists, otherwise some arbitrary, but functionally determined value. If the result is RE, it is replaced by E.

Notice that there is a subtlety in the treatment of E and RE. A receiver that detects a missing or bad value passes it to its interstage as RE, whereas interstages that detect errors pass them on as E.

Notice also that receiving processors do not include the value received directly from the transmitter in their majority vote; instead, they use the value "reflected back" to them by their own interstage. This seems odd, since the interstage could be faulty and so a good value obtained directly from the transmitter will be replaced by a bad value from the faulty interstage. In fact, the chosen arrangement does not reduce fault-tolerance and is actually crucial to correct operation under multiple-fault scenarios.[5]

[5] Consider a quadruplex FTP with one manifest-faulty interstage and a Byzantine-faulty transmitter that sends different values to every receiver and to its interstage; if the processor with the faulty interstage used the value received directly from the transmitter, it would end up with a different set of values than the other nonfaulty processors and, since there will be no overall majority, it could select a different value in the majority vote. We do not know whether or not Draper's original algorithm includes values received directly from the transmitter in each receiver's majority vote.

3.1.5 Informal Proofs of OMH-FTP

An FTP-n architecture running OMH-FTP has the same fault-tolerance as an $n+1$-plex running OMH(1). Let a be the number of arbitrary-faulty processors and interstages, s the number that are symmetric-faulty, and m the number that are manifest-faulty.

Lemma 1 *For any a, s, m, Algorithm OMH-FTP satisfies the Validity requirement if $n > 2(a+s)+m$.*

Proof: Validity only specifies what must happen if the transmitter is not Byzantine-faulty. If the transmitter is nonfaulty or symmetric-faulty, then all nonfaulty receivers obtain the value actually sent by the transmitter (call this ν) in step (1). Those nonfaulty receivers with interstages will forward that value to their interstages in step (2), and it will then be forwarded by all nonfaulty interstages to all processors in step (3). Let m'_i be the actual number of manifest-faulty interstages, and let m'_p be the actual number of manifest-faulty receivers with nonfaulty interstages. Then a nonfaulty receiver receives at least $n-(a+s+m'_i+m'_p)$ values equal to ν, and at least $m'_i+m'_p$ equal to E.[6] Thus, at most $n-(m'_i+m'_p)$ non-E values are received, and ν will be selected in the *H-majority* vote provided

$$2(n-(a+s+m'_i+m'_p)) > n-(m'_i+m'_p),$$

that is, provided

$$n > 2(a+s)+(m'_i+m'_p).$$

Clearly, $m \geq m'_i+m'_p$ and the result follows.

Next, we consider the case where the transmitter is manifest-faulty. In step (1), all the nonfaulty receivers note the value received as RE. We now have to perform a case-analysis depending on whether or not the transmitter has an interstage, and whether or not it is faulty. If the transmitter has no interstage, the analysis is the same as the previous case: a nonfaulty receiver obtains at least $n-(a+s+m'_i+m'_p)$ values equal to RE, and at least $m'_i+m'_p$ equal to E. (Remember, the E values received by receivers directly from the transmitter are transformed into RE before being sent on to their interstages.)

If the transmitter has a nonfaulty interstage, then the transmitter is counted among the m'_p. A nonfaulty receiver therefore receives at least $n-1-(a+s+m'_i+$

[6] There are subcases here, depending on whether or not the transmitter has an interstage, and whether or not the transmitter is nonfaulty or symmetric-faulty; however, the stated bounds are correct for all cases.

as were our early attempts to correct it—and partly because the argument is intrinsically more complicated, due to the extended case analysis needed for the additional fault modes tolerated by the hybrid algorithm OMH(m) [16].

With satisfactory treatments of the Oral Messages Algorithm for the FTP architecture (OM-FTP) and for the hybrid fault model (OMH(m)) available separately, we combined them to yield the OMH-FTP algorithm and its formal verification described in the remainder of this section.

3.1 Informal Treatment of OMH-FTP

A fault model, set of requirements, algorithm, and argument for the algorithm's correctness are presented informally in the following subsections. The formal treatment is described a later section. In the following, the transmitter and the receivers always refer to processors, not interstages.

3.1.1 Fault Model

As noted earlier, our fault model is the same as that of Thambidurai and Park, but with the cases renamed. The fault modes we distinguish are *arbitrary-faulty*, *symmetric-faulty*, and *manifest-faulty*. Of course, we also need a class of *nonfaulty* (sometimes called *good*) processors. We specify these fault modes semiformally as follows (the formal characterizations are presented later).

When a transmitter or interstage p sends some value t to its receivers, the value obtained by a nonfaulty receiver q is:

- t, if p is nonfaulty
- E, if p is manifest-faulty[4]
- Unknown, if p is symmetric-faulty, but all nonfaulty receivers obtain the *same* value,
- Completely unconstrained, if p is arbitrary-faulty.

Similarly, when a processor sends the value t to its interstage, the value received (if the interstage is nonfaulty) is t or E if the processor is nonfaulty or manifest-faulty, respectively, and unconstrained if it is symmetric- or arbitrary-faulty. Note that a symmetric-faulty transmitter may send a *different* value to its interstage than to the receivers, but that it must send the same value to all receivers. Note also that manifest faults must be symmetric, in the sense that if a processor were to "crash" during this protocol (or if some of its outgoing links are broken, or if it is early or late transmitting on some links), it would have to be counted as arbitrary-faulty, since different nonfaulty receivers may obtain different values—even though the values sent are either correct or identifiably bad. Investigating whether this restriction can be relaxed is an interesting area for future research.

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

3.1.2 Other Assumptions

Nonfaulty processors are assumed to execute correctly the algorithm given below; faulty processors and interstages need not. It is because the values sent by faulty processors and interstages may be unrelated to those received that the values received by faulty processors were left unspecified in the previous subsection.

A nonfaulty transmitter must be able to transmit directly to all the nonfaulty processors that are paired with interstages; each nonfaulty processor paired with a nonfaulty interstage must be able to transmit to that interstage; each nonfaulty interstage must be able to transmit to every nonfaulty processor but receives only from the processor with which it is paired.

3.1.3 Requirements

The Agreement property is standard, but specification of the Validity property has to be extended to account for symmetric and manifest faults.

Agreement: All nonfaulty receivers agree on the value ascribed to the transmitter.

Validity: If receiver q is nonfaulty, the value it ascribes to the transmitter is

- The correct value if the transmitter is nonfaulty.
- The value actually sent to all receivers if the transmitter is symmetric-faulty.
- The distinguished value E if the transmitter is manifest-faulty.

2.3.1 Interactive Consistency for FTP

OM-FTP, the interactive consistency algorithm developed at Draper Laboratory for the FTP architecture [12, page 340] is the following variation on the traditional single-round Oral Messages algorithm OM(1) [15]. The transmitter sends its value directly to all the receiving processors, which then forward the value to their own interstages (the transmitter also sends its value to its interstage). Each interstage then sends the value it has received to every processor (including its own). Finally, each processor performs a majority vote on the values receives from the interstages.

Published papers describing FTP [10, 12] do not present a detailed analysis of the fault tolerance achieved by this algorithm, although Lala [12, page 342] states that a quadruplex FTP can tolerate "certain combinations of two processor and two interstage failures."

Our contributions, described in the sections that follow, are the development of an extension to the interactive consistency algorithm for FTP so that not all processors need have interstages, the extension of that algorithm to better account for manifest failures under a hybrid fault model, and formal specification and verification of both algorithms.

3 The OMH-FTP Algorithm and its Formal Specification and Verification

Our development and formal verification of an algorithm for interactive consistency in FTP under a hybrid fault model built on several earlier verifications of related algorithms.

For our first step, we developed a formal specification and verification of the classical Oral Messages algorithm OM(m) for a symmetric architecture using the EHDM verification system [23]. The key insight here (due originally to Bevier and Young [2]) is that although the Oral Messages algorithm is conceived as a distributed algorithm to be executed on loosely synchronized processors using message-based communications, its correctness and fault-masking capabilities can be explored and verified in a model that represents the algorithm as a recursive function in classical logic, with the fault model captured in a function $send(t, p, q)$ that represents the value received by nonfaulty processor q when processor p attempts to send it the value t (if p is nonfaulty, then the received value is t otherwise, under the Byzantine fault model, the value is unspecified).

Our colleague Shankar converted this specification from EHDM [27] to PVS [21] and recast it in the Byzantine Generals (as opposed to Interactive Consistency) formulation. This specification and verification was used as one of the test cases that guided the development of PVS and it was also used as one of our standard demonstrations, so that we became very familiar with the argument.

By specializing the formal specification for OM(m) to the case $m = 1, n = 3$ and adding the interstages and their communications, we were able to construct a specification of OM-FTP for a triplex FTP architecture quite easily. Then, building on our familiarity with the correctness argument for OM(m), we were able to construct mechanically-checked proofs for the correctness of OM-FTP in a couple of hours.

While documenting an informal version of the correctness argument for the triplex OM-FTP, we realized that it was not necessary for every processor in FTP to have a corresponding interstage: as long as there are at least three processor-interstage pairs, additional processors can be accepted without interstages. We therefore extended the triplex FTP specification to encompass n-plex FTP architectures, where $n \geq 3$ is the number of processor-interstage pairs, and any number of additional processors lacking interstages may also be present. We modified the previous triplex proofs using the proof-editing facility of PVS, and in less than a day of extra work we had a complete specification and formal verification of single-fault tolerance of n-plex FTP architectures. The ability to accept processors without interstages is a useful extension: it allows interstages diagnosed as faulty to simply be excluded from the configuration; this could be valuable if certain sensors are directly observed only by specific processors. (Previously, we had assumed that a processor must be dropped from the configuration if its interstage is diagnosed as faulty).[3] Our analysis shows that all processors can still participate as transmitter or receiver without impact on fault tolerance.

At this point, we turned from FTP back to symmetric architectures, but for the case of hybrid fault models. Our formal specification and verification for this version of the problem built directly on our previous treatment of OM(m) but was much more difficult—partly because the published algorithm was incorrect,

[3] We are unsure whether the designers of FTP had already invented this extension; we suspect they had, since they speak of the ability of a quadruplex FTP to tolerate "certain combinations of two processor and two interstage failures" [12, page 342], but we have not found a published analysis of this case.

2.3 The FTP Architecture

Extending the classical algorithms and their analyses from Byzantine to hybrid fault models can be viewed as work aimed at wringing more fault tolerance from a given level of replication. A complementary investigation would seek to minimize the quantity and complexity of hardware needed to achieve a given level of fault tolerance. We say "quantity and complexity of hardware" rather than simply "level of replication" or "number of processors" since the first suggests a better measure of real costs. The minimality results of Pease, Shostak and Lamport [22] show that at least four participants are needed in order to achieve interactive consistency in the presence of a single arbitrary fault. However, although all participants need to be able to receive and transmit messages, not all of them need to be able to vote, and only those that are to participate in other computational activities need to be full processors.

The effects of a single faulty processor on the results of computational activities (e.g., evaluation of control laws) can be masked by three-way majority voting, so only three full processors are required to accomplish the main activity of the system in a fault-tolerant manner. A fourth processor is needed only to ensure single-fault tolerance in the distribution of single-source sensor data (i.e., interactive consistency), and for clock synchronization. The quantity and complexity of hardware required for overall single-fault tolerance could be reduced if the full processor in the fourth processor could be replaced by some minimal component that is just sufficient to discharge the responsibilities of interactive consistency and clock synchronization. This, approximately, is the rationale underlying Draper Laboratory's FTP architectures [10], except that rather than add a minimal fourth processor, these architectures supplement *each* of three processors with an extremely simple component called an "interstage" (see Figure 1).

The FTP architectures have undergone a considerable evolution and simplification over the years [10,12] (for example, early versions performed voting in the interstages and required 50 times as much hardware as the present design [12, page 343]); the final version, also incorporated to some extent in AIPS [13] and FTPP [9], has either three or four processors, and the same number of interstages. The processors are fully connected among themselves, and each processor is connected to every interstage. Interstages can consist of nothing more than message forwarding hardware. (Basically, they operate like mirrors: each reflects data from its host processor back to every processor.) The triplex version of FTP, as shown in Figure 1 has only three processors but six "fault containment regions" (three processors and three interstages) and can withstand a single Byzantine fault in any of the components. The quadruplex version of FTP (four processor-interstage pairs) can withstand any two faults in succession, provided it can reconfigure to exclude the faulty unit between the arrival of the first and second faults.

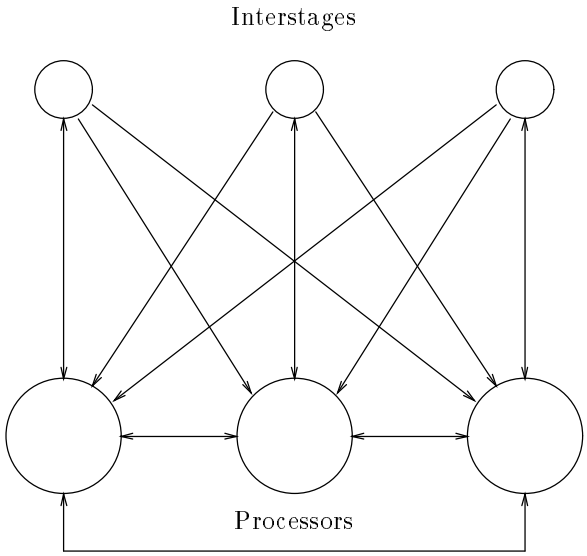

Figure 1: Triplex FTP Architecture

A triplex FTP contains six components, but only three of them are processors. The three interstages are elementary units requiring so few gates that all three together comprise less silicon than a single processor. Thus, the overall hardware complexity, and hence the fault-arrival rate, of a triplex FTP should be less than that of a conventional quadruplex, and its cost and reliability should be correspondingly superior. Similarly, a quadruplex FTP similarly provides comparable fault tolerance to a conventional 5-plex, but with only four full processors. A minor limitation of the FTP architecture is that it does not appear suitable for versions of the Oral Messages Algorithm that use several rounds of information exchange (i.e., OM(m) with $m \geq 2$), and thus cannot be extended to tolerate two or more arbitrary faults simultaneously. We consider this a minor limitation because $m \geq 2$ is seldom used in practice.

The Oral Messages algorithm of Lamport, Shostak, and Pease [15], which we denote OM(m) is a recursive algorithm that uses $m+1$ rounds of message exchange. In the base case, OM(0), each receiver accepts whatever value it obtains from the transmitter; in the general case, OM(m), each receiver takes the part of the transmitter in OM($m-1$) to communicate the value it received from the transmitter to the other recipients, and each receiver settles on the majority value among all those obtained (i.e., one directly from the transmitter and $n-2$ via the other receivers, where n is the total number of processors). OM(m) can withstand up to m arbitrary faults, provided $n > 3m$. The bound $n > 3m$ is optimal: Pease, Shostak, and Lamport proved that no algorithm based on the Oral Messages assumptions (and, implicitly, a symmetric architecture) can withstand more arbitrary faults than this [22]. However, OM(m) is not optimal when special classes of faults are considered: other algorithms can withstand greater numbers of simpler faults for a given number of processors than OM(m). We examine fault models in the next section.

2.2 Hybrid Fault Models

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

Classical Byzantine fault-tolerant algorithms such as Oral Messages make no assumptions about the behavior of faulty components. Their advantage is that they cannot be defeated by unexpected fault modes; their disadvantage is that all faults are treated as "worst case," so that large levels of redundancy tolerate relatively few faults.

These observations motivate the study of fault-tolerant architectures and algorithms with respect to fault models that include arbitrary faults, together with a limited number of additional common fault modes. Inclusion of the arbitrary fault mode eliminates the fear that some unforeseen fault 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.

Thambidurai and Park [29], Meyer and Pradhan [19], and Garay and Perry [7] have considered Interactive Consistency algorithms that resist multiple fault classes. We adopt a *hybrid* fault model identical to Thambidurai and Park's "Unified" model but with the cases renamed (to avoid their anthropomorphic distinction between "malicious" and "nonmalicious" faults). We divide faults into three classes: manifest, symmetric, and arbitrary. A *manifest* fault is one that produces a value that all nonfaulty receivers can detect as bad. Timing, omission and crash faults are in this category. Symmetric and arbitrary faults 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 all nonfaulty receivers. As before, an *arbitrary* fault is completely unconstrained and, in particular, may deliver different wrong (or missing or detectably bad) values to different nonfaulty receivers. (The fault models of Meyer and Pradhan and of Garay and Perry are similar to this but neglect the symmetric case.)

Thambidurai and Park [29] presented a variation on the Oral Messages Algorithm that solves the Byzantine Generals Problem under the hybrid fault model. Unfortunately, this algorithm (though not its implementation in MAFT) is flawed. With the support of the PVS verification system [21], we were able to detect this flaw and to develop a correct version of this algorithm [18] that is able to withstand a arbitrary, s symmetric, and c manifest faults simultaneously, using m rounds, provided there are more than $2a+2s+c+m$ processors and $a \leq m$. We also constructed a formal, mechanically-checked proof of the correctness of this algorithm [16]. The essence of this enhanced algorithm, which we call OMH(m), is that it recognizes manifest faulty values and excludes them from the majority votes (the flaw in Thambidurai and Park's algorithm, which is repaired in our version, is that it is necessary to treat manifest faulty values specially when communicating them in the recursive instances of the algorithm).

Thambidurai, Park, and Trivedi [30] present reliability analyses that show that this increased fault-tolerance does provide significantly superior reliability under plausible assumptions. Notice that when only arbitrary faults are present, we have $a = m$ and $s = c = 0$, so that $n > 3m$ as with the classical Oral Messages Algorithm.

hybrid fault model was developed at the Aerospace Technology Center of Allied Signal for their MAFT ("Multicomputer Architecture for Fault Tolerance") architecture [11]. Prototypes were constructed for both architectures, and they and their successors are being considered, evaluated, or used for safety and control applications in nuclear plants, aircraft, helicopters, submarines, and rockets.

Although the architecture, algorithm, and fault model investigated here are interesting and useful in their own right, we are equally interested in the general use of mechanically-checked formal methods as a systematic and rigorous means to analyze critical algorithms, to identify all the assumptions on which they depend, to detect and help correct errors in their formulation, and to provide compelling arguments for their correctness. From this point of view, algorithms for fault tolerance, especially those analyzed under hybrid fault models, are particularly interesting because of their criticality, subtlety, and the extended case analysis required in their study. However these complex combinations of fault-tolerant architectures, algorithms, and fault models are well within the scope of practical machine-checked formal verification as we show below.

2 Interactive Consistency, Hybrid Fault Models, and the FTP Architecture

In this section we review the problems of Interactive Consistency and Byzantine Agreement, and describe the Oral Messages algorithm. We then introduce hybrid fault models and the FTP architecture. None of this material is new and the whole section can be skipped by those familiar with these topics.

2.1 Interactive Consistency

In this paper, we focus on algorithms for reliably distributing single-source data to multiple processors in the presence of faults. This problem, known as "Interactive Consistency" (although sometimes called "source congruence" or "consensus"), was first posed and solved for the case where faulty processors can exhibit arbitrary behavior by Pease, Shostak, and Lamport [22] in 1980. Interactive Consistency is a symmetric problem: it is assumed that each processor has a "private value" (e.g., a set of sensor samples) and the goal is to ensure that every nonfaulty processor achieves an accurate record of the private value of every other nonfaulty processor. In 1982, Lamport, Shostak, and Pease [15] 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 processor called the "Commanding General" to all the other processors, 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. An algorithm for the Byzantine Generals problem can be converted to one for Interactive Consistency by simply iterating it over all processors (each processor in turn taking the role of the Commander), so there is no disadvantage to considering the Byzantine Generals formulation. See [23] for more extended discussion of this topic.

In its Byzantine Generals formulation, the problem is to communicate a value from a "transmitter" processor to several other "receiver" processors in such a way that the following two properties are satisfied.

Agreement: All nonfaulty receivers agree on the value obtained from the transmitter.

Validity: If the transmitter is nonfaulty, then every nonfaulty receiver obtains the correct value.

Note that here and henceforth, we use the terms *transmitter* and *receivers* for the metaphorical "Commanding General" and "Lieutenant Generals."

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

that only $2m+1$ processors are required to withstand m faults [24]. The additional processors required for clock synchronization and interactive consistency not only increase cost and complexity, they also increase the fault arrival rate (since this may be expected to grow in proportion to the amount of hardware employed) and thereby possibly reduce overall reliability.

- Although 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, they are not maximally effective with respect to the *number* of faults they can tolerate: other algorithms (e.g., simple majority voting) can withstand more faults of particular kinds (e.g., crash-faults) for a given level of redundancy than traditional Byzantine-resilient algorithms. However, these alternative algorithms may fail when confronted by faults beyond the kinds they are designed to handle.

Two recent developments mitigate these disadvantages.

Asymmetric Architectures. SIFT was a symmetrical design: all its processors were identical. Pease, Shostak and Lamport [22] proved that four such symmetric processors are required to withstand a single (arbitrary) fault. The C. S. Draper Laboratory's *Fault Tolerant Processor* (FTP) is an asymmetrical design consisting of conventional processors and much simpler *interstages* [12]. The interstages merely relay messages; they have no computational capability and do not participate in the voting steps that provide fault-tolerance in FTP. Arbitrary single-fault tolerance is achieved with only three processors and three interstages, so the FTP architecture ought to be cheaper and more reliable than a symmetrical quadruplex system providing the same fault tolerance. Since FTP contains only three processors capable of voting, it appears to violate Pease, Shostak and Lamport's minimality result [22] and the intuition that "four voters are required to resist one Byzantine fault." Consequently, very careful justification is required for its correctness and fault tolerance.

Hybrid Fault Models. All faults were treated as Byzantine in SIFT; for simple classes of faults, this treatment fails to extract the maximum fault-tolerance from a given level of redundancy. For example, a five-processor Byzantine-resilient system using a traditional interactive consistency algorithm can withstand a single arbitrary fault (without reconfiguration), but may fail if two faults arrive simultaneously, even if those faults are of a simple character that could be masked by ordinary majority voting. Thambidurai and Park [29], Meyer and Pradhan [19], and Garay and Perry [7] have developed fault models that include certain common and simple kinds of faults in addition to arbitrary faults. Algorithms developed for these fault models can tolerate as many arbitrary faults as a Byzantine fault-tolerant algorithm, but can also tolerate more simple faults and combinations of faults.

The subtlety and criticality of the algorithms and implementation strategies employed in fault-tolerant systems argue for the use of formal methods as a means of design assurance. A research program led by NASA Langley Research Center [4] has precisely this goal. So far the classical Byzantine fault tolerant clock synchronization algorithms [26,28], the Oral Messages Algorithm for Interactive Consistency [2,23], fault masking and transient recovery by majority voting [6,24], extensions of clock synchronization to hybrid fault models [25] and transient recovery [20], extensions of the Oral Messages algorithm to the hybrid fault model [16,18], and several levels in the implementations of these algorithms [5] have been subjected to mechanically-checked formal specification and verification.

In this paper, we describe the formal verification of an "Oral Messages" algorithm for achieving Interactive Consistency in the asymmetric FTP architecture under a hybrid fault model. This combination of architecture, algorithm, and fault model is the best compromise that we know between economy and fault tolerance for this problem: other combinations either tolerate fewer faults or less severe kinds of faults for a given level of redundancy, or they require more hardware to tolerate the same number and kinds of faults.[2]

Both novel elements of this work have industrial, rather than academic, origins: the FTP architecture comes from the C. S. Draper Laboratories [12], and the

[2] Algorithms based on "signed messages" [22] can tolerate more faults than those based on oral messages (such as the algorithm considered here). To do so, however, they require more "rounds" of information exchange, and rest on strong assumptions concerning the authentication methods used. We have recently investigated algorithms that use authentication to enhance the fault tolerance of oral messages algorithms, without making assumptions about the effectiveness of authentication when within the competence of the basic algorithm.

Formal Verification of an Interactive Consistency Algorithm for the Draper FTP Architecture Under a Hybrid Fault Model*

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

Abstract

Fault-tolerant systems for critical applications should tolerate as many kinds *of faults and as large a* number *of faults as possible, while using as little hardware as feasible. And they should be provided with strong assurances for their correctness.*

Byzantine fault-tolerant architectures are attractive because they tolerate any kind fault, but they are rather expensive: at least $3m + 1$ *processors are required to withstand* m *arbitrary faults. Two recent developments mitigate some of the costs: algorithms that operate under a* hybrid fault model *tolerate more faults for a given number of processors than classical Byzantine fault-tolerant algorithms, and* asymmetric architectures *tolerate a given number of faults with less hardware than conventional architectures. In this paper we combine these two developments and present an algorithm for achieving interactive consistency (the problem of distributing sensor samples consistently in the presence of faults) under a hybrid fault model on an asymmetric architecture.*

The extended fault model and asymmetric architecture complicate the arguments for the correctness and the number of faults tolerated by the algorithm. To increase assurance, we have formally verified these properties and checked the proofs mechanically using the PVS verification system. We argue that mechanically-supported formal methods allow for effective reuse of intellectual resources, such as specifications and proofs, and that exercises such as this can now be performed very economically.

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

1 Introduction

The general design outline of a "reliable computing platform" for ultra-critical applications was established in the late 1970s and early 1980s by the SIFT architecture [8,31] and later refined in the FTP [12] and MAFT [11] architectures: the system workload is executed by several independent processors in approximate synchrony, and the results are subjected to exact-match majority voting. Clock synchronization, and also the distribution of single-source data such as sensor samples, is performed in a manner that is resistant to arbitrary (or "Byzantine") faults [14,22].[1] The great advantage of a "Byzantine-resilient" design such as this is that its overall reliability is dependent only on the fault arrival rate, the degree of replication, and the correctness of its design and implementation; it does not depend on identifying and countering all the individual fault modes that can afflict the underlying hardware.

However, a disadvantage of this approach is that it requires a lot of hardware to withstand relatively few faults: both clock synchronization and interactive consistency (the problem of distributing sensor samples consistently in the presence of faults) require at least $3m+1$ processors to withstand m arbitrary faults. This is unfortunate on at least two grounds.

- Apart from clock synchronization and interactive consistency, fault tolerance in SIFT-like architectures is achieved by simple majority voting, so

[1] Technically, a Byzantine fault is one that is entirely unconstrained: no assumptions are made about the behavior of Byzantine-faulty components. We generally prefer to call this the *arbitrary* fault mode, since Byzantine faults are popularly interpreted as "malicious" faults that display asymmetric symptoms (e.g., that communicate different values to different recipients)—even though this is only one of the possible manifestations of an arbitrary fault.