SRI International

CSL Technical Report • September 2001

Formal Verification of Hybrid Byzantine Agreement Under Link Faults

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

This is a draft of work in progress. It is made available for information and to solicit feedback. Please do not distribute it without the author's permission.

This research was supported by NASA Langley Research Center under contract NAS1-20334 and Cooperative Agreement NCC-1-377 with Honeywell Tucson.

Computer Science Laboratory • 333 Ravenswood Ave. • Menlo Park, CA 94025 • (650) 326-6200 • Facsimile: (650) 859-2844

Abstract

We describe an extended fault model for Byzantine Agreement due to Schmid and Weiss. The new fault model extends the previous "hybrid" fault model of Thambidurai and Park by the addition of omission-faulty nodes, and by the introduction of link faults. We formally verify the Hybrid Oral Messages Algorithm (OMH) under this new fault model.

Our formal verification improves the analysis of Schmid and Weiss a little by increasing the independence of node and link faults, and by making explicit the exact assumptions required in final rounds and in the case OMH(0). We also refute a conjecture of Schmid and Weiss that the analysis can be extended to omission-faulty receivers and introduce a symmetric version of the algorithm that does handle these cases correctly.

Contents

Chapter 1

Introduction

Safety-critical systems may replicate applications for enhanced reliability. Replicas may be designed to operate either in primary/monitor, or primary-backup, or modular-redundant (i.e., voted) configurations and may either be "diverse" (i.e., different replicas use different algorithms or implementations, and may operate on different data) or identical to one another. Diversity is controversial, but is favored by some as a way to protect against design errors. Diverse replicas cannot be expected to produce identical results, so pairwise comparisons and voting must use approximate-match algorithms (all of which have problems [Dav87]). Identical replicas executing on different processors are used to tolerate random hardware faults: here it is crucial that all replicas operate on the same data so that comparisons and voting can use exact-match algorithms. The problem of ensuring that different replicas operate on the same data (and hence maintain identical state) is variously referred to as "consensus," "interactive consistency," "replica determinism," or "Byzantine agreement." In this report, we examine and formally verify an algorithm for Byzantine Agreement that is is resilient to a wide class of faults.

The basic problem to be addressed is that of communicating data values consistently from one replica to several others. Simply broadcasting the data from the sending host may not be adequate for required reliabilities in the 10^{-9} range: the sending host could be faulty, as could be any of the receivers, and the communication mechanisms might drop or corrupt messages. To overcome these potential faults in the most robust manner, it is necessary to cross-check the data received at each node. The basic method for doing this is the so-called "Oral Messages" algorithm for Byzantine Agreement [LSP82]. Most systems are designed to a single-fault assumption, so the "one round" version of Oral Messages is adequate. This algorithm operates as follows:

1. The sending node broadcasts its value to all receiving nodes.
2. Each receiving node broadcasts the value received to all other receivers.
3. Each receiving node performs a majority vote on the values received.

Provided there are three or more receivers, this algorithm tolerates any single fault and achieves the following two properties.

Agreement: all nonfaulty receivers agree on the same value.

Validity: if the sender is nonfaulty, the value agreed by nonfaulty receivers is the value actually sent.

A weakness in this algorithm is that it is not guaranteed to tolerate any faults when there are fewer than three receivers, and it is not guaranteed to tolerate more than a single fault no matter how many receivers there are.

To improve matters, we need a more elaborate "fault model" that distinguishes different kinds and severities of faults, and a slightly different algorithm that retains the ability to tolerate a single arbitrary fault, but works better when several simple faults are present. A suitable fault model is the "Hybrid" model (introduced by Thambidurai and Park of Allied Signal [TP88, TPT89]) that distinguishes three kinds of fault: *manifest*, *symmetric*, and *arbitrary*, defined as follows.

When a node sends a value v to others, the value obtained by a nonfaulty recipient is

- v, if the sender is nonfaulty
- E, if the sender is manifest-faulty
- Unknown, if the sender is symmetric-faulty, but all nonfaulty recipients obtain the *same* value
- Completely unconstrained, if the sender is arbitrary-faulty

The idea is that E represents a special "error" value that is used to indicate that a receiver detected something wrong: either no value was received, or it failed to checksum, or was detectably bad in some other way.

A simple modification to the Oral Messages algorithm allows it to tolerate more simple (i.e., manifest, and symmetric) faults: simply discard E values from the majority votes. This modification was introduced by Thambidurai and Park, who gave it the name "Algorithm Z." Unfortunately, it fails in certain circumstances (e.g., a manifest-faulty sender and one arbitrary-faulty receiver). This deficiency was noted and corrected by Lincoln and Rushby, who called their corrected algorithm "Oral Messages Hybrid" (OMH) [LR93b]. This algorithm "wraps" values in some function R before relaying them from one receiver to another. The wrapper is removed (by applying an "unwrap" function *UnR*) following the majority votes. OMH overcomes the bug in Algorithm Z by allowing error values (E) to be distinguished from the *reports* of such values ($R(E)$). Lincoln and Rushby formally verified the general (r-round) case for OMH using PVS and proved that the algorithm can tolerate a arbitrary, s symmetric, and m manifest faults simultaneously, provided $r \geq a$ and $n > 2a + 2s + m$, where n is the number of nodes [LR93a]. Thus, one-round OMH on three nodes can tolerate a single manifest fault (it can actually tolerate two, but the problem is vacuous in that case), and on five nodes it can tolerate an arbitrary and a manifest fault simultaneously, and so on.

A useful extension to this fault model, introduced by Schmid and Weiss [SW01] is the *omission* fault: this can be thought of as a restricted arbitrary fault or as a less restricted manifest fault. When a node sends a value v to others, the value obtained by a nonfaulty recipient is

- Either v or E if the sender is omission-faulty

It is fairly easy to see that omission faults are similar to manifest faults in the difficulty they present to OMH in achieving Validity. Thus, the algorithm can tolerate a arbitrary, s symmetric, o omission, and c manifest faults simultaneously, provided $r \geq a$ and $n > 2a + 2s + o + m$, where n is the number of nodes. However, these faults are as difficult as arbitrary faults in achieving Agreement so we need $r \geq a + o$. Schmid and Weiss also claim that the Validity and Agreement properties can be extended from nonfaulty receivers to omission-faulty receivers.

A weakness in OMH is that it does not allow for *link* faults: these are faults in which a broadcast from one node is missed or received incorrectly by some nodes but correctly received by others. Schmid and Weiss' introduction of omission-faults goes some way towards repairing this, but it still requires that we attribute the faults to a node (we must count the sender as arbitrary-faulty or omission-faulty, or must count the receivers that did not get the correct value as symmetric-faulty). If several nodes suffer just a few incoming or outgoing link faults, we will soon use up our "budget" of arbitrary- or omission-faulty nodes, and yet it is fairly easy to construct scenarios in which the algorithm does work correctly in the presence of several such faults. For example, OMH works correctly in many circumstances in which each receiver misses one incoming message—but under the traditional analysis we would have to count each receiver as faulty and would not be able to make any statement about the behavior of the algorithm in this circumstance. Thus, the current analysis for OMH potentially underestimates the true fault tolerance of this algorithm.

Recently, Schmid and Weiss [SW01] have extended analysis of OMH and related algorithms to include link faults. They allow link faults to change the value received, as well as simply to lose the value, so that a link fault has some of the complexity of an arbitrary fault. The obvious extension, where a bound is placed on the number of receivers that may incorrectly receive a given broadcast, does not work because the same receiver may receive several broadcasts incorrectly in the same execution of the algorithm, and its majority vote may then select an incorrect value. The clever innovation of Schmid and Weiss is to bound not only the number of "outgoing" broadcasts that can be received incorrectly, but also the number of "incoming" broadcasts.

In the following Chapter, we outline the analysis of Schmid and Weiss and subject it to formal verification.

Chapter 2

Formal Verification of Byzantine Agreement

We begin by presenting a traditional mathematical proof of the correctness of OMH(1) in the presence of both node and link faults. This proof is based on a modification of that for the general r-round case developed by Schmid and Weiss [SW01]. Following this, we discuss the general case and describe its formal verification in PVS.

We assume n participants, which we call "nodes." A distinguished node, which we call the *sender*, possesses a value to be communicated to all the other nodes, which we call the *receivers*. The Byzantine Agreement problem can be studied under several different assumptions: we use the "Oral Messages" assumptions, and a "Hybrid" fault model extended with omission faults and link faults.

The *Oral Messages* assumptions are the following [LSP82, p. 387].

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

A2: The receiver of a message knows who sent it (assumption of private channels).

A3: The absence of a message can be detected (assumption of synchrony).

Under our fault model, link faults can change or lose messages between good nodes, so assumption A1 is dropped.

There are n nodes in total, of which some (possibly including the sender) may be faulty. In the classical Byzantine Generals formulation of the problem, there are no constraints other than those given above on the behavior of faulty nodes. This leads to pessimistic estimates of the number of faults that can be tolerated, because all faults are regarded as the worst possible. We consider a "hybrid" fault model that distinguishes certain simpler kinds of fault as well as those that are unconstrained. The fault modes we distinguish for nodes are *arbitrary-faulty*, *omission-faulty*, *symmetric-faulty*, and *manifest-faulty*. The intuition behind these fault modes was described in Chapter 1. We also allow link faults, and these may change the values received by nonfaulty recipients of messages sent by nonfaulty senders. For this reason, we do not characterize the values exchanged between individual nodes in this case, but only those between an individual sender and a *set* of recipients, or an individual receiver and a *set* of senders.

Fault Model

The fault model distinguishes "node" faults from "link" faults; the latter are further divided into "send" and "receive" faults.

Node Faults: Manifest, omission, and arbitrary faults are attributed to the sending node.

- When a manifest-faulty node sends a value, all nonfaulty receivers obtain the value E.
- When an omission-faulty node sends a value v, nonfaulty receivers obtain either the value v or the value E. (Note: one receiver may obtain v and another E.)
- When a symmetric-faulty node sends a value v, all nonfaulty receivers obtain the value $perturb(v)$. We call $perturb(v)$ the value *actually sent*.
- When an arbitrary-faulty node sends a value, there are no constraints on the value received.

Link Send Faults: These concern nonfaulty senders.

- When a nonfaulty node sends its value v to a set of nonfaulty and omission-faulty receivers, at most ls of them receive a value different from v.

Link Receive Faults: These concern only nonfaulty senders.

- When a set of nonfaulty nodes sends values to a nonfaulty receiver, at most lr of the values received may differ from those sent; of those, at most lra of the values received may be different from E (the E values correspond to messages that are dropped).

Note that $lr = 0$ iff $ls = 0$.

Requirements

Under these assumptions, the Byzantine Agreement problem is to devise an algorithm that allows each receiver p to compute an estimate ν_p of the sender's value v satisfying the following conditions.

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

Validity: If the sender is not arbitrary-faulty and the receiver p is nonfaulty, the value ascribed to the sender by p is

- The correct value if the sender is nonfaulty—that is, $\nu_p = v$
- The value actually sent, if the sender is symmetric-faulty—that is, $\nu_p = perturb(v)$
- The value E, if the sender is manifest-faulty—that is $\nu_p = E$
- Either the correct value, or the value E, if the sender is omission-faulty—that is $\nu_p = v \vee \nu_p = E$.

Algorithm

The algorithm is OMH(1); it proceeds in three steps.

1. The sender sends its value to all receivers.
2. Each receiver applies the function R to the value received from the sender and sends this value to all other receivers (including itself).
3. Each receiver then selects the majority among those received in Step 2, with E values excluded. If there is no absolute majority, the value $R(E)$ is used.[1] Each receiver then applies UnR to the value selected.

Correctness

Let a, s, o and m be the number of arbitrary-faulty, symmetric-faulty, omission-faulty, and manifest-faulty nodes, respectively.

- OMH(1) achieves validity provided

$$n > 2(a + s + ls) + lr + lra + o + m + 1.$$

- OMH(1) achieves agreement provided

$$n > 2(a + s + ls) + lr + lra + o + m + 1$$

and

$$a + o = 0 \vee (a + o = 1 \wedge ls = 0).$$

[1]Notice that this is different from the majority function used in the original OMH for the basic hybrid fault model: in that case, the majority function returns some arbitrary, but functionally determined value when there is no absolute majority [LR93b]. The adjustment to return $R(E)$ is necessary to accommodate omission faults.

Proof

We first consider validity in the case that the sender is nonfaulty.

Let k be the number of nonfaulty and omission-faulty nodes among the receivers. Then $k - ls'$ of these obtain the value v in Step 1 of the algorithm, where ls' is the number of outgoing link faults in the sender's transmission. Let x be the number of nonfaulty nodes among the $k - ls'$ that correctly receive v, and let o' be the corresponding number of omission-faulty nodes. Note that $x + o' = k - ls'$.

Then any nonfaulty receiver p obtains at least $x - lr'$ values equal to $R(v)$ from nonfaulty receivers in Step 2 of the algorithm, where lr' is the number of link faults incoming to p from nonfaulty receivers that received the correct value in Step 1. In addition, p receives $o' - oo'$ values equal to $R(v)$ from omission-faulty receivers in Step 2 of the algorithm, where oo' is the number of omission-faulty nodes among the $k - ls'$ receivers that received the correct value in Step 1, but delivered E to p (so the number that correctly delivered $R(v)$ is $o' - oo'$). Thus, p receives $R(v)$ from at least $x - lr' + o' - oo'$ receivers in Step 2. Recall that $x + o' = k - ls'$, so the number of receivers delivering $R(v)$ to p can be written as $k - ls' - lr' - oo'$.

Let $lr' = lra' + lro'$ where lro' is the number of incoming link faults to p from nonfaulty receivers that received the correct value in Step 1, but delivered E to p in Step 2, and lra' is the number that delivered something other than the correct value or E. Also, let m' be the number of manifest-faulty nodes among the $n - 1$ receivers. Then the nonfaulty receiver p obtains at least $oo' + m' + lro'$ Es in Step 2 of the algorithm, and therefore most $n - 1 - oo' - m' - lro'$ values different from E.

It follows that $R(v)$ will win the majority vote in Step 3 of the algorithm provided

$$2(k - ls' - lr' - oo') > n - 1 - oo' - m' - lro'.$$

Now $k = n - 1 - a' - s' - m'$ where a' and s' are the numbers of arbitrary-faulty and symmetric-faulty receivers, respectively. Thus, we need

$$2(n - 1 - a' - s' - m' - ls' - lr' - oo') > n - 1 - oo' - m' - lro',$$

or equivalently

$$n > 2(a' + s' + ls' + lr') - lro' + oo' + m' + 1.$$

We have $a \geq a'$, $s \geq s'$, $o' \geq oo'$, $m \geq m'$, $ls \geq ls'$, $lr' - lro' = lra'$, $lra \geq lra'$, and $lr \geq lr'$, and so the constraint follows from that stated in the correctness condition.

Validity in the other cases follows by similar arguments. A symmetric-faulty sender is like a nonfaulty one that sends $perturb(v)$ instead of v, while a manifest-faulty sender is like a nonfaulty one that sends E. For an omission-faulty sender, the argument is again similar, except that the constraint ensures that the majority of values in the vote are equal either to $R(E)$ or to $R(v)$. Notice that this means that no other value can be in the majority. If either of these values *individually* has a majority, then that value will be selected; otherwise (if there is no majority) the value $R(E)$ will be selected by default.

Agreement follows from validity in cases where the sender is neither arbitrary- nor omission-faulty. When the sender is arbitrary- or omission-faulty, we must have $ls = 0$ (and hence $lr = 0$), and none of the receivers can be arbitrary-faulty. It follows that each receiver obtains exactly the same set of values from the other receivers in Step 2 of the algorithm and decides the same value in the majority vote (since this is functionally determined). $\square$

This proof, and our formulation of the fault model, differ from those of Schmid and Weiss in a couple of ways. First, link send faults may afflict omission-faulty receivers as well as nonfaulty ones. This means that our calculation of the number of values correctly received is performed differently than that of Schmid and Weiss (though the result is the same). Second, contrary to the description given in the fault model, we implicitly assumed that link send faults may affect faulty as well as nonfaulty senders. Examination of the proof indicates that this is sound. The two changes have the benefit of increasing the independence between node and link faults.

The phrase "examination of the proof" suggests that human review of the proof presented above is both feasible and reliable. However, there are many cases to consider: the sender may be correct or in any of four fault modes, these are compounded by link faults, and there are two properties to consider (validity and agreement). Furthermore, Schmid and Weiss observe that the analysis should be extended so that validity and agreement are shown to hold for omission-faulty receivers—thereby increasing the case analysis by another factor of two. It is difficult for human reviewers to examine each of these many cases with the independent scrutiny that is really required (recall the phrase "follows by similar arguments" in the current proof), and this raises some doubts about the reliability of human review for these kinds of proofs. Although a reader may be persuaded that the algorithm works as advertised, this may owe more to intuitive understanding than to a conviction that the formal details of the proof are correct. It is in these circumstances that mechanically-checked formal verification can be most valuable—especially when we wish to explore weakened fault assumptions (such as those that allow link faults to afflict faulty nodes), since these generally increase the amount of case analysis that is required.

The proof for the general r-round algorithm is similar to, but more difficult than, that presented above for $r = 1$. The proof is by induction on r but, because its constraints are rather different from the other cases, it is simplest to verify the $r = 0$ case separately, and to organize the induction with $r = 1$ (i.e., the proof presented above) as the base case. The proof for the inductive step is then similar to the $r = 1$ case, except that the values voted in the final step have already been subjected to a recursive instance of the algorithm and this allows weaker fault assumptions than those required in Step 2 of the OMH(1) algorithm (for example, link receive faults need be constrained by lr only in the final round).

We describe the formal verification in PVS of the r-round algorithm in the following section.

2.1 Formal Verification in PVS

The formal verification in PVS is developed in a theory `genlinks` (for "generalized link faults") parameterized by a positive number `n` that is the number of participants (or nodes), and a nonempty type `T` that is the domain of values over which the agreement algorithm is to operate. We then define `node` as the type $0 \ldots n-1$ that represents the nodes. We then import the PVS library theories for finite sets and define `nodeset` as the type of finite sets of nodes and `vector` as the type of functions from nodes to `T`.

```
genlinks[n: posnat, T: NONEMPTY_TYPE]: THEORY
BEGIN

npos: LEMMA n > 0
node: TYPE+ = below[n] CONTAINING 0

IMPORTING cardprops[node],
  finite_sets@finite_sets_below,
  finite_sets@finite_sets_pred

nodeset: TYPE = finite_set[node]
vector: TYPE = [node -> T]
```

We next introduce `E` as a distinguished constant of type `T` and `R` and `UnR` as functions on `T` and specify three axioms that characterize their properties. We also use another distinguished constant `arbval` of type `T`, but leave its value entirely unconstrained.

```
E: T
UnR, R:[T->T]

R_ax: AXIOM UnR(R(t)) = t
R_E_prop: AXIOM R(t)/=E
UnR_inv: AXIOM UnR(x)=t => x=R(t)

arbval: T
```

We use the uninterpreted function `send(r, t, p, q)` to represent the value received by node `q` when node `p` attempts to send it the value `t` in round r. The fault model will be specified later as a series of constraints on the behavior of `send`. We use the uninterpreted function `maj(caucus, v)` to represent the result of a majority vote of the values in the vector `v`, restricted to those belonging to nodes in the set `caucus`. Its properties will be specified later. Given these two functions, we can specify the Oral Messages algorithm as the recursive function OMH.

```
send(r, t, p, q): T

maj(caucus, v): T

OMH(G, r, t, caucus)(p) : RECURSIVE T =
  IF r = 0
  THEN IF member(G, caucus) AND member(p, caucus)
       THEN send(0, t, G, p)
       ELSE arbval
       ENDIF
  ELSE IF member(G, caucus) AND member(p, caucus)
       THEN IF p = G
           THEN send(r, t, G, G)
           ELSE
               UnR(maj(remove(G, caucus),
                  (LAMBDA q: OMH(q, r - 1, R(send(r, t, G, q)),
                                 remove(G, caucus))(p))))
           ENDIF
       ELSE arbval
       ENDIF
  ENDIF
MEASURE r
```

`OMH(G, r, t, caucus)(p)` represents the value received by node `p` when node `G` distributes the value `t` to the set of nodes `caucus` using the `r`-round Hybrid Oral Messages algorithm. If either `G` or `p` is not a member of `caucus`, then the value is unspecified (we use `arbval` for this purpose). Otherwise, if `r = 0`, or `G = p`, `G` sends `t` directly to `p`. Otherwise, `G` sends `t` to all nodes `q` in `caucus` (other than itself); these nodes then use OMH on `r-1` rounds to distribute the wrapped versions of values they receive to each other: the expression

```
LAMBDA q: OMH(q, r-1, R(send(r,t,G,q)), remove(G, caucus))(p)
```

is a function that specifies the value `p` receives from each receiver `q` as a result of the inner call to `OMH(r-1)`. Finally, `p` takes the majority among those values received and unwraps the result.

Observe that round numbers (which appear, for example, in the invocation of `send`) count down from `r` to `0` as the recursion proceeds. A round with number 0 is the most deeply nested, or "final" round of the algorithm.

The kinds of faults considered are enumerated in the type `faults`, and the uninterpreted function `status(p)` specifies the kind of fault that afflicts node `p` (the `good` fault means "nonfaulty"). We then introduce predicates `ok`, `man` etc. as convenient abbreviations (so that `ok(p)`, for example, is true when the `status` of `p` is `good`).

```
faults: TYPE = {good, manifest, omission, symmetric, arbitrary}

status(p): faults

ok(p): bool = good?(status(p))
man(p): bool = manifest?(status(p))
omit(p): bool = omission?(status(p))
sym(p): bool = symmetric?(status(p))
arb(p): bool = arbitrary?(status(p))
```

The fault model is specified as a series of constraints on behavior of `send`. Many of these constraints use a predicate `val(p, t, x)` which indicates whether reception of `x` is consistent with `p` sending `t` in the absence of link faults; this predicate is defined by cases on the fault status of `p`: we need a predicate rather than a function because of the nondeterministic behavior of omission-faulty nodes.

```
perturb(p, t): T

val(p, t, x): bool =
 CASES status(p) of
    good: x=t,
    manifest: x=E,
    omission: x=t or x=E,
    symmetric: x=perturb(p,t),
    arbitrary: true
  ENDCASES
```

The function `perturb(p, t)` models the "value actually sent" when symmetric faulty node `p` attempts to send `t`.

It might be expected that we would state the general axiom `val(p, t, send(r, t, p, q))`, but this would neglect the possibility of link faults. Therefore, we define `als(caucus, G, j, t, p)` as the cardinality of the set of nonfaulty and omission-faulty receivers in the set `caucus` that get a value *not* consistent with the fault mode of `G` when `G` sends `t` to all the members of `caucus`: this is the number of link send faults. We then assert that this is bounded by `ls` when the round number is nonzero.

```
okm(q): bool = ok(q) OR omit(q)

als(caucus, G, j, t): nat =
  card({q | member(q, remove(G, caucus)) AND okm(q)
             AND not val(G, t, send(j,t,G,q))})

als_ax: AXIOM j>0 => als(caucus, G, j, t) <= ls
```

Notice that this is a liberalization of the fault model of Schmid and Weiss in that link send faults are allowed to afflict any kind of sender, not just nonfaulty ones, and apply to omission-faulty receivers as well as nonfaulty ones (we do not care about the values received by manifest-, symmetric-, or arbitrary-faulty receivers since the values these send need not be related to those that they receive).

Memo: May be better to remove $j > 0$ from `als_ax`; then the three axioms below become lemmas.

Notice that we restrict `als_ax` to the case `j` $>$ `0`; the reason is that the case `j = 0` arises in two different circumstances and we wish to characterize the fault assumptions as precisely as possible in each circumstance. The first circumstance is where the top-level invocation is OMH(0): here the sender simply broadcasts to the receivers and we cannot tolerate any link faults (i.e., we will need `ls = 0`). When `ls = 0` and `j` $>$ `0`, it is a provable consequence of `als_ax` that `val(G, t, send(j,t,G,q))`; we specify that this applies also in the case `j = 0` when the receiver is nonfaulty, and when the receiver is omission-faulty and the sender is either nonfaulty or symmetric-faulty. We will need these properties also when the sender is manifest- or omission-faulty, but these constraints also apply when `ls` $>$ `0` and are stated later.

```
ls0: AXIOM ls=0 => ok(q) => val(G, t, send(0,t,G,q))

s_send_ax: AXIOM
           ls=0 => sym(G) and omit(q) => val(G,t,send(0,t,G,q))

o_rcv_ax0: AXIOM
           ls=0 => ok(G) and omit(q) => val(G,t,send(0,t,G,q))
```

The other circumstance arises when the top-level invocation is OMH(r) for some `r` $>$ `0`. Here, multiple instances of OMH(0) occur: in particular, each node in the parent OMH(1) round will use OMH(0) (i.e., `send`) to convey its value to node `p`. Node `p` will be able to tolerate the loss or corruption of some of these transmissions, and we specify this in terms of the function `alr` and its variants, and the constraint `alr_lim`.

```
alr(caucus, v, p): nat =
    card({q | member(q,caucus) AND ok(q)
               AND send(0, v(q), q, p) /= v(q)})
alro(caucus, G, v, p): nat =
    card({q | member(q, caucus) AND ok(q)
               AND send(0, v(q), q, p) = E})
alra(caucus, G, v, p): nat =
    card({q | member(q, caucus) AND ok(q)
               AND send(0, v(q), q, p) /= v(q)
               AND send(0, v(q), q, p) /= E})

alr_lim: AXIOM
   alr(caucus, v, p) <= lr and alra(caucus, v, p) <= lra
```

The function `alr(caucus, v, p)` specifies the number of receive link faults suffered by `p` when each member `q` of some set `caucus` sends its value `v(q)` to `p` in a final round. The functions `alro` and `alra` are similar and give the number of omission link faults, and non omission link faults, respectively. It is a fairly obvious lemma that the first of these function equals the sum of the other two.

```
alr_prop: LEMMA (FORALL q: v(q)/=E) =>
      alr(caucus, v, p) = alro(caucus, v, p) + alra(caucus, v, p)
```

Manifest- and omission-faulty senders must not suffer link faults in final rounds when the receiver is either nonfaulty or omission-faulty.

```
m_send_ax: AXIOM  man(G) AND okm(q) => val(G,t,send(0,t,G,q))

o_send_ax: AXIOM omit(G) AND okm(q) => val(G,t,send(0,t,G,q))
```

Link faults only afflict transmissions between different nodes: the sender always receives its own value correctly (provided it is not arbitrary faulty).

```
send_self_ax: AXIOM not arb(p) => val(p,t,send(r,t,p,p))
```

It remains only to specify the majority function. The basic specification is that if we have a set of nodes `caucus`, and two subsets `ayes` and `Es`, where all the `ayes` have some value `x` and all the `Es` have the value `E`, then `x` will win the majority vote provided

```
2 × |ayes| > |caucus| - |Es|
```

(i.e., the `ayes` form a strict majority of the non`E` values).

The actual specification is more complicated than this because we need to deal with omission-faulty senders, where either a wrapped ordinary value or `R(E)` could be in the majority (and `R(E)` is also the default value when there is no majority). We therefore restate the specification of majority in terms of values satisfying some predicate `Q` rather than a single value `x`. For this to be sound, we require that `Q` is not satisfied by `E` and is satisfied only by some single value, or by `R(E)`.

```
Q: VAR [T->bool]

hybrid_maj_ax: AXIOM
 (FORALL (x:node): member(x,caucus) and Q(v(x)) => v(x)/=E)
  AND (FORALL (x, y:node): member(x,caucus) AND Q(v(x))
                         AND member(y,caucus) AND Q(v(y))
        => v(x)=v(y) or v(x)=R(E) OR v(y)=R(E))
   AND (EXISTS (ayes, Es): subset?(ayes, caucus)
     AND subset?(Es, caucus)
      AND (FORALL p: member(p, ayes) => Q(v(p)))
       AND (FORALL p:  member(p, Es) => v(p) = E)
        AND 2 * card(ayes) > card(caucus) - card(Es))
 IMPLIES Q(maj(caucus, v))
```

We also require that the majority is functionally determined by the members of `caucus` (this is needed when there is no majority).

```
maj_ext: AXIOM
  FORALL (v1, v2: vector):
    (FORALL p: member(p, caucus) => v1(p) = v2(p))
      IMPLIES maj(caucus, v1) = maj(caucus, v2)
```

With these definitions and axioms, we can develop the lemmas that establish the correctness of the OMH algorithm in the presence of link faults. The first of these establishes Validity for the case `r = 0`. In this case, we require `ls = 0 AND lr = 0`, and the result follows by straightforward reasoning and use of the `ls0` axiom.

```
validity0: LEMMA
     ok(p) AND member(p, caucus) AND member(G, caucus)
       AND ls=0 AND lr=0
         IMPLIES val(G, t, OMH(G, 0, t, caucus)(p))
```

The next series of lemmas verifies Validity for the case `r = 1`; this is the base case for the induction that establishes the result for general `r`. We begin with a lemma that establishes a lower bound on the number of nodes `q` that correctly relay to nonfaulty or omission-faulty receiver `p` the value received when a non arbitrary-faulty sender `G` uses `OMH(1)` to send the value `t`.

```
count_t1: LEMMA
  okm(p) AND NOT arb(G) AND member(p, remove(G,caucus))
    IMPLIES
      card({q | Let x = send(1, t, G, q) IN
                member(q, remove(G, caucus))
                 AND val(G, t, x)
                  AND OMH(q, 0, R(x), remove(G, caucus))(p) =  R(x)})
          >= akm(remove(G,caucus))
                 - als(caucus,G, 1,t)
                 - alr(remove(G,caucus),lambda q:R(send(1,t,G,q)),p)
                 - aoo(remove(G,caucus),G, 0,t,p)
```

Here, we use akm(caucus) to count the number of nonfaulty and omission-faulty nodes in the set caucus. Similar definitions are used to count faulty nodes of different kinds.

```
akm(caucus): nat = card({q | member(q,caucus) AND okm(q)})
aa(caucus): nat = card({q | member(q,caucus) AND arb(q)})
as(caucus): nat = card({q | member(q,caucus) AND sym(q)})
am(caucus): nat = card({q | member(q,caucus) AND man(q)})
ao(caucus): nat = card({q | member(q,caucus) AND omit(q)})
```

We use the function aoo to count the number of omission-faulty nodes that deliver E, rather than the correct value, to p when G uses OMH(j) to send the value t.

```
aoo(caucus,G,j,t,p): nat =
  card({q | member(q,caucus) AND omit(q)
       AND OMH(q,j, R(send(1+j,t,G,q)), caucus)(p) = E})
```

The proof of count_t1 uses the following result to split the cardinality expression that appears in its statement into a sum over the corresponding expression for each kind of node q.

```
count_status: LEMMA card(A) =
   card({q | A(q) AND man(q)})
     + card({q | A(q) AND ok(q)})
     + card({q | A(q) AND omit(q)})
     + card({q | A(q) AND arb(q)})
     + card({q | A(q) AND sym(q)})
```

The proof of count_t1 uses the following result to rearrange the cardinality expressions, and then establishes inequalities between the resulting terms and those appearing in the lemma.

```
  card_prop: SUBLEMMA
    card({q:T | A(q) AND B(q)}) =
      card({q:T | A(q)}) - card({q:T | A(q) AND NOT B(q)})
```

The next lemma establishes a lower bound on the number of `E` values that are received by a nonfaulty or omission-faulty receiver `p` when a non arbitrary-faulty sender `G` uses `OMH(1)` to send the value `t`.

```
count_E1: LEMMA
  okm(p) AND member(p,remove(G,caucus))
   implies
     card({q | member(q, remove(G, caucus)) AND
          OMH(q, 0, R(send(1,t,G,q)), remove(G, caucus))(p) = E})
        >= am(remove(G,caucus))
           + aoo(remove(G,caucus),G,0,t,p)
           + alro(remove(G,caucus),lambda q:R(send(1,t,G,q)),p)
```

Proof of this lemma uses `count_status` and the axiom `m_send_ax`.

The main result for the `r = 1` case is then stated as follows.

```
validity1: lemma
  NOT arb(G) AND okm(p) AND member(p, caucus)
    AND member(G, caucus)
     AND card(caucus) > 2*(aa(caucus) + as(caucus) + ls)
           + am(caucus) + ao(caucus) + lr + lra + 1
         IMPLIES val(G, t, OMH(G, 1, t, caucus)(p))
```

Proof of this result uses `hybrid_maj_ax` and appeals to the `count_t1` and `count_E1` lemmas, together with the following axioms: `R_E_prop`, `R_ax`, `UnR_inv`, `alr_lim`, `als_ax`, `o_send_ax`, and `send_self_ax`.

Next, we establish lemmas for the general case. The first is similar to `count_t1` and is proved in a similar way. Observe, however, that unlike `count_t1`, the constraint here does not mention `alr`.

```
count_t: LEMMA
   okm(p) AND NOT arb(G) AND
     (FORALL q,x: okm(q) AND member(q,remove(G,caucus)) IMPLIES
         val(q, R(x),  OMH(q, j, R(x), remove(G, caucus))(p)))
    IMPLIES
      card({q | LET x = send(1 + j, t, G, q) IN
           member(q, remove(G, caucus))
            AND val(G, t, x)
             AND OMH(q, j, R(x), remove(G, caucus))(p) =  R(x)})
        >= akm(remove(G,caucus))
           - als(caucus, G, j+1, t)
           - aoo(remove(G,caucus), G, j, t, p)
```

Next, `count_E` is the corresponding generalization of `count_E1`. Observe, however, that `alro` is not used in this constraint.

```
count_E: LEMMA okm(p) and
  (FORALL q, x: man(q) AND member(q,remove(G,caucus)) IMPLIES
         OMH(q, j, x, remove(G, caucus))(p) =  E)
   IMPLIES
     card({q | member(q, remove(G, caucus)) AND
      OMH(q, j, R(send(1+j,t,G,q)), remove(G, caucus))(p) = E})
        >= am(remove(G,caucus))
           + aoo(remove(G,caucus),G,j,t,p)
```

Finally, `validity` states the general property for $\texttt{r} \geq 1$. It is proved by induction, with the base case supplied by `validity1`.

```
validity: LEMMA  FORALL (r:upfrom(1)):
  NOT arb(G) AND okm(p)
    AND member(p, caucus) AND member(G, caucus)
     AND card(caucus)> 2*(aa(caucus) + as(caucus) + ls)
           + am(caucus) + ao(caucus) + lr + lra + r
         IMPLIES val(G,t,OMH(G, r, t, caucus)(p))
```

The proof depends on the following axioms: `R_E_prop`, `R_ax`, `UnR_inv`, `alr_lim`, `als_ax`, `hybrid_maj_ax`, `m_send_ax`, `o_send_ax`, and `send_self_ax`.

Given `validity`, it is quite simple to establish `agreement`. The proof is by induction on `r`, starting from 0.

```
agreement: LEMMA
  member(G,caucus)
   AND ok(p) AND member(p, caucus)
    AND ok(q) AND member(q, caucus)
      AND card(caucus) > 2*(aa(caucus) + as(caucus) + ls)
          + am(caucus) + ao(caucus) + lr + lra + r
     AND r > aa(caucus) + ao(caucus) + min(1,ls)
       IMPLIES OMH(G, r, t, caucus)(p) = OMH(G, r, t, caucus)(q)
```

When the sender `G` is arbitrary- or omission-faulty, the result follows from `maj_ext`, and in other cases it follows from `validity`.

Schmid and Weiss observe that the `Validity` and `Agreement` properties can and should be extended from nonfaulty receivers to those that are omission-faulty. The rationale is that omission-faulty receivers must be executing the basic algorithm correctly to be able sometimes to pass on the correct value: their main purpose is to model nonfaulty nodes with "lots" of link faults (more than `ls`). Accordingly, we extended and generalized some of the previous results. This was done silently for `validity1` and `validity`, where it will be observed that the results are stated for receivers `p` such that `okm(p)` (i.e., `p` is either nonfaulty or omission-faulty). The one result that needs a somewhat different proof for the omission-faulty case is validity for `r = 0`.

```
validity0o: LEMMA
     omit(p) AND member(p, caucus) AND member(G, caucus)
       AND ls=0 and lr=0
         IMPLIES val(G, t, OMH(G, 0, t, caucus)(p))
```

The proof depends on the axioms `m_send_ax`, `o_send_ax`, `s_send_ax`, `send_self_ax`, and `o_rcv_ax0`.

When we attempted to generalize Agreement to omission-faulty nodes, however, our proof attempts failed, leading us to discover that this property can fail in this case. The counterexample is an omission-faulty sender that attempts to send `t` but actually sends only `E` values. All nonfaulty or omission-faulty receivers will decide on `E`, but the sender will have sent the value `t` to itself correctly, and will thereby violate the Agreement property. The significance of this discovery, and other observations on the formal verification, are presented in the following chapter.

Chapter 3

Discussion

The formal verification described in the previous chapter is quite a difficult exercise. Its development took about two weeks of full-time effort, and a similar amount of part-time preparation and experiment. Some of this difficulty can be attributed to the complexity of reasoning about cardinalities of finite sets. Often, one has two set-valued expressions, say `A` and `B`, and wishes to establish that they have equal cardinalities by observing that they represent the same set. The general way to do this is by application of the extensionality axiom, but this is rather tedious and heavy-handed when the expressions `A` and `B` appear identical to the naked eye. Similar frustrations arise where one wishes to replace instances of the left hand side of the set equality `A=B` by its right hand side: one can often see instances of `A` within the sequent, but PVS refuses to replace them by `B`. The source of both these frustrations can be found by causing PVS to fully parenthesize expressions in proof sequents: it will be seen that apparently identical conjunctive expressions appearing in set comprehensions are actually associated differently. Adding the following lemma as an automatic rewrite rule normalizes these expressions in right-associative form and substantially increases the automation of proofs in this domain.

```
r,s,t: VAR bool

norm_assoc: SUBLEMMA ((r AND s) AND t) = (r AND (s AND t))
```

Note that application of this rewrite rule inside set comprehensions depends on PVS' ability to perform automatic rewriting under bound variables. Further improvement in mechanization of proofs involving cardinality may be possible through recent work by Klaedtke and Rueß: motivated by this example, they have extended the decidable logic WS1S to include certain formulas involving cardinality [KR01].

The main source of difficulty in these proofs, however, is not due to difficulties in mechanization but to the numerous cases that must be considered and the delicacy with which assumptions must be stated and matched up across various lemmas and axioms. These difficulties are equally pronounced in informal proofs, and tend to discourage attempts to discover the most general formulation of a given result, or the minimal axiomatization necessary to support it. Once a mechanized formal proof has been developed, however, it is relatively straightforward to experiment with generalizations and alternative formulations, and the results are completely reliable as the mechanized theorem prover examines each variation with the same untiring scepticism.

This largely unrecognized benefit of formal verification—its ability to serve as an aid in design exploration—allowed us to generalize Schmid and Weiss' formulation so that node and link faults are largely independent of each other: in our formulation, link send faults are allowed to afflict faulty as well as nonfaulty senders, and omission-faulty as well as nonfaulty receivers. In addition, we are able to demarcate precisely the assumptions required in nonfinal rounds vs. those required in final rounds, and can distinguish the assumptions required for OMH(0) from those required in the final rounds of OMH(r).

Our first verification followed the semi-formal proof of Schmid and Weiss. Here, the constraint on the number of send link faults applies only to nonfaulty receivers; the proof implicitly assumes that link send faults do not afflict omission-faulty receivers. The formal verification required explicit axiomatization of this implicit assumption, and thereby exposed its unreasonable character. This led to the reformulation used here, in which the constraint on the number of link send faults applies to both nonfaulty and omission-faulty receivers, and the proof was modified correspondingly.

A more widely recognized benefit of formal verification—its ability to detect flaws in designs or arguments—led us to discover the flaw in Schmid and Weiss' conjecture that the correctness results for OMH(r) can be extended from nonfaulty to omission-faulty receivers. This discovery has ramifications beyond this particular example and points to a deficiency in all previous treatments of hybrid fault-tolerant consensus algorithms, and indicates the need to revise the algorithms as well as their analyses.

In its original formulations [LSP82, PSL80], consensus was to be used in architectures that employ n-modular redundancy and voting to mask processor faults. In this context, values held by faulty processors are irrelevant because they will be masked by voting: all that matters is that the nonfaulty processors agree and outnumber the faulty ones. Furthermore, faulty processors were assumed to be Byzantine (i.e., arbitrary-faulty), so it made little sense to consider their behavior. Under the hybrid fault model, however, several less-than-arbitrary fault modes are distinguished, and it is plausible to attribute these to processors that are working correctly, but have some bad values in their state (leading, for example, to off-by-one errors that may be manifested as symmetric faults) or faulty communications (leading to manifest or omission faults). In these circumstances, we may be interested in the values of faulty processors—because this may allow less pessimistic calculations for fault masking, or may allow different strategies for fault tolerance. One such strategy, called *compensation* [Rus01a, Rus01b] is contemplated for many drive-by-wire applications. Under a compensation strategy, different processors have access to different actuators: none is a direct backup to any other, but each changes its operation when it deduces that others have failed. Car braking provides an example: separate processors controlling the braking force at each wheel will redistribute the force when one of their number has failed. The safety of this strategy depends crucially on the expectation that a failed processor leaves its actuator in a predictable state (e.g., applying zero braking force). Consider now a manifest-faulty processor. All nonfaulty processors will determine that the value ascribed to this processor is E. If the value represents the braking force to be applied, those nonfaulty processors will assume that the faulty processor has failed and is applying no force to its wheel. But under OMH(1), the faulty processor would have no reason to suspect its own faultiness and, using the `send_self_ax` axiom, would see its intended braking force as t and would apply that force—with possibly disastrous consequences. This scenario is simply a variation on the one we discovered as a counterexample to Schmid and Weiss' conjecture that OMH ensures agreement for omission-faulty processors. It points to the conclusion that in some contexts, such as the fault compensation strategy, it can be important for faulty processors to participate in the agreement property.

The reason for the failure of Agreement in the examples considered is the asymmetry between sender and receivers: the sender does not participate in the additional rounds of exchange and voting performed by the receivers. If we eliminate this asymmetry, as in the following version of OMH(1), we can show that faulty senders do participate correctly in the Agreement property, under suitable assumptions about their behavior.

Revised (symmetric) version of OMH(1).

1. The sender sends its value to all processors (including itself).
2. Each processor applies the function R to the value received from the sender and sends this value to all processors (including itself).
3. Each processor then selects the majority among those received in Step 2, with E values excluded. If there is no absolute majority, the value $R(E)$ is used. Each processor then applies UnR to the value selected.

In the braking scenario described above, the manifest-faulty sender would send `R(t)` to itself in step 2, but receive `R(E)` from all other processors; like them, it would then select `E` as the value attributed to itself.

It can be seen that this symmetric revision to OMH(1) incorporates some of the properties of group membership within consensus. A dual approach is to incorporate some of the properties of consensus within group membership. This is what is actually done in the Time Triggered Architecture (TTA) [KG94], whose transmissions can be considered to comprise an authenticated variant on OMH(0) (authenticated because the CRC check used in TTA acts like a digital signature). Authentication should eliminate arbitrary- and symmetric-faulty behaviors and transform them into manifest- or omission-faults. This will result in (at most) two cliques: those that receive a real value and those that receive `E`. The group membership and clique-avoidance algorithms of TTA [BP00] will then eliminate whichever of these cliques is the smaller, so that the survivors all satisfy the Agreement property. This method is referred to as Draconian Consensus [Rus01b]. Interesting challenges for the future will be to formally verify the symmetric revision to OMH(1) and the Draconian Consensus of TTA, and to compare them.[1] Draconian consensus is employed in the standard transmissions of TTA, whereas the variant of OMH(1) is proposed for the FT CNI and for the distribution of sensor readings.[2]

Other interesting opportunities for the future include formal verification of the "authenticated" variants to OMH in the presence of link faults. These algorithms, called OMHA and ZA, were introduced by Gong, Lincoln, and Rushby [GLR95]. They have an advantage over the classical "signed messages" algorithm of Lamport, Pease, and Shostak [LSP82] in that failure of the authentication assumption does not render them useless (they simply revert to their behavior in the unauthenticated case) yet they are able to exploit authentication to increase the number of faults that can be tolerated (ZA achieves the same fault-tolerance as the signed messages algorithm in this case). Schmid and Weiss have extended analysis of these algorithms to include link faults [SW01], and it should be straightforward to extend the formal verification of OMH in the presence of link faults to the algorithms OMHA and ZA in a similar manner.

Finally, we note that Schmid has developed a treatment of clock synchronization [Sch01] under a fault model that includes link faults and we believe that this will provide an improved foundation for formal verification of the TTA clock synchronization algorithm.

[1] Draconian consensus requires formal verification of the clique-avoidance component of the TTA group membership algorithm. The core agreement algorithm has already been formally verified by Pfeifer [Pfe00].

[2] We have presented OMH in its Byzantine Agreement formulation; the form anticipated for TTA is the symmetric Interactive Consistency formulation; the two formulations are interdefinable [Rus92].

Bibliography

[BP00] Günther Bauer and Michael Paulitsch. An investigation of membership and clique avoidance in TTP/C. In *19th Symposium on Reliable Distributed Systems*, Nuremberg, Germany, October 2000.

[Dav87] Gloria J. Davis. An analysis of redundancy management algorithms for asynchronous fault tolerant control systems. NASA Technical Memorandum 100007, NASA Ames Research Center, Moffett Field, CA, 1987.

[FTC95] IEEE Computer Society. *Fault Tolerant Computing Symposium 25: Highlights from 25 Years*, Pasadena, CA, June 1995. IEEE Computer Society.

[GLR95] Li Gong, Patrick Lincoln, and John Rushby. Byzantine agreement with authentication: Observations and applications in tolerating hybrid and link faults. In Ravishankar K. Iyer, Michele Morganti, W. Kent Fuchs, and Virgil Gligor, editors, *Dependable Computing for Critical Applications—5*, volume 10 of *Dependable Computing and Fault Tolerant Systems*, pages 139–157, Champaign, IL, September 1995. IEEE Computer Society.

[KG94] Hermann Kopetz and Günter Grünsteidl. TTP—a protocol for fault-tolerant real-time systems. *IEEE Computer*, 27(1):14–23, January 1994.

[KR01] Felix Klaedtke and Harald Rueß. WS1S with cardinality constraints. Technical report, Computer Science Laboratory, SRI International, Menlo Park, CA, 2001. to appear.

[LR93a] Patrick Lincoln and John Rushby. Formal verification of an algorithm for interactive consistency under a hybrid fault model. In Costas Courcoubetis, editor, *Computer-Aided Verification, CAV '93*, volume 697 of *Lecture Notes in Computer Science*, pages 292–304, Elounda, Greece, June/July 1993. Springer-Verlag.

[LR93b] Patrick Lincoln and John Rushby. A formally verified algorithm for interactive consistency under a hybrid fault model. In *Fault Tolerant Computing Sympo-*

sium 23, pages 402–411, Toulouse, France, June 1993. IEEE Computer Society. Reprinted in [FTC95, pp. 438–447].

[LSP82] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine Generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, July 1982.

[Pfe00] Holger Pfeifer. Formal verification of the TTA group membership algorithm. In Tommaso Bolognesi and Diego Latella, editors, *Formal Description Techniques and Protocol Specification, Testing and Verification FORTE XIII/PSTV XX 2000*, pages 3–18, Pisa, Italy, October 2000. Kluwer Academic Publishers.

[PSL80] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2):228–234, April 1980.

[Rus92] 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.

[Rus01a] John Rushby. Bus architectures for safety-critical embedded systems. In Tom Henzinger and Christoph Kirsch, editors, *EMSOFT 2001: Proceedings of the First Workshop on Embedded Software*, Lecture Notes in Computer Science, Lake Tahoe, CA, October 2001. Springer-Verlag. To appear.

[Rus01b] John Rushby. A comparison of bus architectures for safety-critical embedded systems. Technical report, Computer Science Laboratory, SRI International, Menlo Park, CA, September 2001. Available at `http://www.csl.sri.com/~rushby/abstracts/buscompare`.

[Sch01] Ulrich Schmid. How to model link failures: A perception-based fault model. In *The International Conference on Dependable Systems and Networks*, pages 57–66, Goteborg, Sweden, July 2001. IEEE Computer Society.

[SW01] Ulrich Schmid and Bettina Weiss. Consensus with oral/written messages: Link faults revisited. Technical report, Technical University of Vienna, Department of Automation, February 2001. (Draft).

[TP88] 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.

[TPT89] 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.