Corrected and slightly expanded version of a paper presented at 11th International Workshop on Distributed Algorithms (WDAG '97), Saarbrücken Germany, September 1997. Springer-Verlag Lecture Notes in Computer Science Vol. 1320, pp. 155–169.

Low-overhead Time-Triggered Group Membership[*]

Shmuel Katz[1,2], Pat Lincoln[1], and John Rushby[1]

[1] Computer Science Laboratory, SRI International, Menlo Park, CA 94025 USA
email: {lincoln, rushby}@csl.sri.com
[2] Computer Science Department, The Technion, Haifa, Israel
email: katz@cs.technion.ac.il

Keywords: time-triggered protocol, group membership, synchronous algorithms, fault tolerance, formal modeling

Abstract. A group membership protocol is presented and proven correct for a synchronous time-triggered model of computation with processors in a ring that broadcast in turn. The protocol, derived from one used for critical control functions in automobiles, accepts a very restrictive fault model to achieve low overhead and requires only one bit of membership information piggybacked on regular broadcasts. Given its strong fault model, the protocol guarantees that a faulty processor will be promptly diagnosed and removed from the agreed group of processors, and will also diagnose itself as faulty. The protocol is correct under a fault-arrival assumption that new faults arrive at least $n+1$ time units apart, when there are n processors. Exploiting this assumption leads to unusual real-time reasoning in the correctness proof.

1 Introduction and Motivation

Group membership has become an important abstraction in providing fault-tolerant services for distributed systems [2]. As in any protocol for group membership, the one presented here allows nonfaulty processors to agree on the membership, and to exclude apparently faulty ones. Because of the strong fault model used, the protocol we consider has the additional desirable properties that the nonfaulty processors agree on the membership at *every* synchronous step, only faulty ones will be removed from the membership, and removal will be prompt. Moreover, a processor with a fault will also diagnose itself promptly.

This protocol for group membership is appropriate for bandwidth-constrained broadcast networks because it requires only one acknowledgment bit to be piggybacked onto existing regularly scheduled broadcasts. The protocol is derived

[*] This work was supported by Arpa through USAF Electronic Systems Center Contract F19628-96-C-0006, by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under contract F49620-95-C0044, and by the National Science Foundation under contract CCR-9509931.

from one in a tightly integrated protocol architecture for automobile control [9]. Our contribution is to isolate this group membership protocol (which has not been described explicitly in previous papers), to abstract it from other elements of the integrated protocol, to give a precise formulation of its fault model, and to provide a systematic proof of its correctness. The argument for correctness is interesting and surprisingly intricate because the paucity of information carried in each individual broadcast requires inferences to be made over *sequences* of broadcasts; this, in turn, requires the statement of correctness to be strengthened significantly in order to obtain one that is inductive.

1.1 Background

Algorithms for industrial applications are optimized to deliver maximum utility from minimum resources. These optimizations pose interesting problems in protocol design and analysis that differ from those traditionally considered in the algorithms literature. For example, industrial algorithms for distributed consensus are less concerned with asymptotic reductions in the number of rounds than in maximizing the number of faults that can be tolerated with a small fixed number of rounds (generally two). This leads, for example, to "hybrid" fault models and associated algorithms that permit finer distinctions among faults than purely Byzantine fault models and provide strictly superior fault tolerance [4, 10, 12, 14, 17].

The starting point for the algorithm considered here is the time-triggered protocol (TTP) of Kopetz and Grunsteidl [9]. This protocol is intended for the control of critical functions in automobiles, where communications bandwidth is severely limited, some functions (e.g., ignition timing) require service at very high rates and with great temporal precision, and many functions (e.g., brake-by-wire, steer-by-wire) are safety critical [7]. For these reasons, TTP (and protocols for similar applications, such as ARINC 659 which provides the safety-critical backplane for the top-level avionics functions of the Boeing 777 [1]) are highly integrated, and services such as clock-synchronization, reliable broadcast, group membership, and primary-backup shadowing are combined with the basic data-communication service rather than layered. This allows high-quality services to be provided with very high performance at low overhead (for example, ARINC 659 achieves clock synchronization to within two bit-times at 30 MHz). These protocols also separate fault-tolerance for common cases from those for more severe ones. For example, the group membership protocol of TTP assumes only a single transmit or receive fault (and those faults are themselves narrowly defined) within any two rounds, with more severe fault modes and higher fault arrival rates being handled by a "blackout" operating mode. Empirical data supports these design decisions [9].

Bandwidth is a precious commodity in applications of the family of protocols we study here. Practical considerations such as cost of copper wire, likelihood of failures of interconnects, and lack of skilled maintenance drive designers to focus on simple and cheap hardware interconnect technology such as twisted pair. Extra runs of wire back and forth around a vehicle for redundancy and extra bandwidth are perceived to be too costly. Wireless communication is viewed as impractical due to the extreme interference expected in the environment. Thus relatively low bandwidth is one of the critical concerns of the designers of these protocols. Even an extra bit per message is considered significant in this domain.

The design constraints on a group membership protocol for an application such as TTP are that it should provide timely and accurate identification and exclusion of faulty processors with minimum overhead. The integrated nature of the protocol means that rather than interpose special "group membership packets" into the communications stream, it should piggyback what it needs for group membership onto the regular data packets. One way to do this is for each processor to append its assessment of the current membership to each packet that it sends. Under suitable assumptions, a protocol can be based on this approach [8], but it is clearly expensive—requiring n bits of membership data appended to each broadcast, for an n processor system (n is 10–20 for these applications). Later descriptions of TTP show only two bits being used for this purpose (actually, they show four, but that appears to be due to the fact that the buses are paired) [9], but the membership protocol is not described. In the following sections, we present a protocol that satisfies these constraints, using only one bit per broadcast, and analyze its properties.

In the following section the model, including its fault assumptions, is first described independently of the group membership problem. Then the assumptions that involve group membership are given, and the kind of reasoning needed for proving correctness is described. The detailed protocol is seen in Section 3 while the proof of correctness is given in Section 4. In the final sections we present a justification for the $n + 1$ limit on fault arrivals, sketch extensions to allow repaired processors to rejoin the group, and briefly describe our use of formal analysis with the Murϕ state exploration system.

The paper shows that a level of abstraction familiar to researchers in distributed programming can be used to isolate and reason about one of a suite of protocols that are combined at the implementation level for efficiency reasons. The separation leads to fault assumptions that seem strong, but are complemented by other assumptions and interleaved protocols.

2 The Model

There are n processors (numbered $0, \ldots, n-1$) arranged in a logical ring and attached to a broadcast bus. Execution is synchronous, with a notional *time* variable increased by one at each step; this, in turn, defines a *slot* in the range $0, \ldots, (n-1)$ as *time* mod n. Nonfaulty processors broadcast whenever it is their slot.

The goal of group membership is to maintain a consistent record of those processors that appear able to communicate reliably and to execute the protocol. A group membership protocol need not tolerate all the types of faults that may afflict the system of which it is a part: other protocols, logically both "above" and "below" group membership, handle some types of faults. In TTP, for example, replication of the broadcast buses, and strong CRCs (checksums), effectively eliminate message corruption and reduce message loss to a very low level. Clock synchronization ensures that all nonfaulty processors share a common notion of time, and "bus guardians" with independent knowledge of the broadcast schedule prevent faulty processors from speaking out of turn. State-machine replication [16] or pairwise comparison is used to mask or to detect processor faults. These considerations (and empirical measurements) justify considering only two types of faults in the context assumed here.

Send fault: a processor fails to broadcast when its slot is reached.
Receive fault: a processor fails to receive a broadcast.

As noted above, other types of faults can be ignored because they are separately detected by other elements of the total protocol suite and then manifest themselves as either send or receive faults. For example, a transient internal data fault can lead to a processor shutting down and thus exhibiting a send fault when its slot is next reached.

Observe that a send fault can only occur to a processor when it is in the broadcast slot, and a receive fault can only occur to a processor different from the broadcaster. Notice, too, that messages cannot be corrupted, and that a send fault is consistent: *no* processor receives a message from a send-faulty broadcaster. Faults are intermittent: a faulty processor may operate correctly in some steps and manifest its fault in others. A processor is nonfaulty until it manifests a fault, thereafter it is considered faulty; a processor is *actively* faulty at a step if it manifests its fault at that step. That is, a processor is actively send-faulty at a step if it is expected to broadcast but fails to do so; it is actively receive-faulty at a step if it fails to receive the broadcast from a nonfaulty broadcaster.

Two additional assumptions are crucial to the correctness of our protocol, and are justified by the division between a "blackout" operating mode (not considered here) for coping with massive or clustered failures, and the "normal" mode (implemented by the mechanisms described here) that is required to cope only with relatively sparse fault arrivals.

Fault arrival rate: only one nonfaulty processor becomes faulty in any $n + 1$ consecutive slots.
Minimum nonfaulty processors: there are always at least two nonfaulty processors.

The fault model described so far is independent of the problem of group membership. Now we turn to the aspects needed to specify and describe the group membership protocol.

- Each processor has a *local membership set*, that initially contains all processors.
- Processor q is *expected* (to broadcast) by processor p if the current slot is q, and p's local membership set contains q.

A processor will normally broadcast in its slot; it can never broadcast out-of-turn, but it may fail to broadcast in its slot for two reasons:

– It suffers a send fault in that slot,
– It has diagnosed that it suffered an earlier (send or receive) fault and remains silent to inform other processors of that fact.

Using the assumptions and definitions of this model, it is now possible to summarize the requirements specification for the group membership protocol. The required safety property is that the local membership sets of nonfaulty processors are identical in every step, and contain all nonfaulty processors. Additionally, a progress property is needed to exclude trivial solutions: a faulty processor will be removed from the local membership sets of nonfaulty processors no later than the step following its next broadcast slot.[3] Our protocol also ensures that a faulty processor will eventually remove itself from its own membership set (self-diagnosis).

When a processor does broadcast, it appends an "ack" bit to whatever data constitutes the broadcast. This bit indicates whether or not that processor retained the previous expected broadcaster in its membership set. By observing the presence or absence of expected broadcasts, and by comparing the ack bits of received broadcasts to their own observations, processors are able to diagnose their own and other processors' faults and to maintain consistent membership sets.

Non-receipt of an expected broadcast can leave ambiguous the question of whether the transmitter or receiver is faulty. The report (encoded in the ack bit) from the next expected processor is needed to resolve this ambiguity; this report must be reliable, so we will need to show that the next expected processor must be nonfaulty in this case. This does not follow trivially from the fault arrival rate assumption because, for example, the initial non-receipt of a broadcast could be due to that broadcaster falling silent after self-diagnosing a much earlier receive fault. We will need to establish that the diagnosis of faults is sufficiently prompt that it combines with the fault arrival rate assumption to guarantee that the next expected broadcaster cannot be faulty in these circumstances. Thus certain progress properties cannot be separated from the basic agreement properties in the correctness proof.

3 The Protocol

Processors follow a simple fixed procedure: if it is processor p's slot, and p is in its own local membership set, then p attempts to broadcast. (If p is nonfaulty or receive-faulty, it succeeds; if send-faulty, it does not, but is unaware of the fault.) The broadcast includes one bit of state information defined below: the **ack** bit

[3] Technically, the real-time requirement seen here is a safety property and not a progress (or liveness) property in the sense of [11]. However, it does serve to guarantee that needed steps occur and so we refer to it informally as a progress property.

of the broadcaster. Each other processor updates its own local membership set by applying certain rules (described below) to the bit received (or not) from the expected broadcaster. The rules allow each processor to retain or remove either the expected broadcaster or itself from its local membership set, but it will not change its record of membership for any other processor.

Each processor p uses the global variable **time**, a local constant **slot**, and local variables **membership** and **ack**.

- The global variable **time** is an abstraction justified by clock synchronization among local clocks. As noted in the introduction, clock synchronization is assumed to be part of the complete protocol suite along with group membership, and guarantees that all processors agree on the (discrete) value of **time**.
- **slot** is a natural number in the range $0 \ldots n-1$ that records the position of p with respect to other processors in the order of broadcast. This value is fixed and unique for each processor.
- **membership** is the set of processors in p's current view of the group.
- **ack** is a boolean recording whether p received the previous expected broadcast and agreed with the **ack** bit carried in that broadcast. As will be seen shortly, this means that the **ack** bit is true iff p has retained the previous expected broadcaster in its membership, or p was that broadcaster.

We use **ack**(p) to indicate the **ack** bit of processor p, and **slot**(p) to indicate its slot value. Initially, each processor's **membership** contains all other processors, its **ack** is *true*, the global **time** is some initial value (perhaps 0), and each processor is nonfaulty.

The protocol proceeds by discrete time steps; at each step, one processor may broadcast. That broadcaster is the processor b for which $\textbf{slot}(b) = \textbf{time} \bmod n$. The broadcast contains the broadcaster's **ack** bit, plus any data that may be needed for other purposes. The broadcast will be attempted only if b is in its own membership set, and will succeed only if b is not actively send-faulty in that step.

The protocol is described by specifying how each processor p updates its local variables **membership** and **ack** in terms of their previous values, the receipt or non-receipt of an expected broadcast, and the value of the **ack** bit carried in that broadcast.

We first define the auxiliary predicate $arrived(b, p)$ as *true* in a step if and only if processor p receives a broadcast from b, and b is the expected broadcaster in that step. This predicate can be considered local to p because that processor can sense the non-receipt of a broadcast.

- For each processor p, if the current broadcaster b is not an element of p's **membership**, none of the local variables are changed.
- If p is the broadcaster b and is in its own **membership** set, it broadcasts **ack**(b) and then updates **ack**(b) to *true*.

– Otherwise, when p is not the broadcaster b, each field of p is updated as follows (notice that $\mathbf{ack}(p)$ is a local variable of p, and that $\mathbf{ack}(b)$ is provided in the broadcast received from b).
  - Updated **membership:** same as previous **membership** except possibly for p and b.
    - p is excluded in two cases:
      (a) (NOT $arrived(b,p)$) AND NOT $\mathbf{ack}(p)$, or
      (b) $arrived(b,p)$ AND $\mathbf{ack}(b)$ AND NOT $\mathbf{ack}(p)$.
    - b is excluded in the two cases:
      (c) NOT $arrived(b,p)$, or
      (d) $\mathbf{ack}(p)$ AND NOT $\mathbf{ack}(b)$.[4]
  - Updated **ack:** set to $arrived(b,p)$ AND ($\mathbf{ack}(b)$ OR NOT $\mathbf{ack}(p)$).
    Observe that the updated value of $\mathbf{ack}(p)$ is *true* iff p retains b in its local membership (i.e., it is the negation of the disjunction of (c) and (d)). We say that the broadcast by b is *acceptable* to p if the updated value of $\mathbf{ack}(p)$ is *true*.

Thus, p removes itself if (a) two consecutive expected broadcasts are unacceptable, or (b) it considers the previous broadcast unacceptable, but b considers it acceptable. Moreover, p removes b if (c) no broadcast is received or (d) p considers the previous expected broadcast acceptable, while b does not.

The broadcaster always assumes that its broadcast was correctly received even when that was not the case, and thus it sets its **ack** bit to *true*. For other processors, the **ack** bit will be *true* in the following step exactly when the broadcast arrives and either the broadcaster views the previous expected broadcast as acceptable, or the receiver does not.

4 Proof of Correctness

The key safety property of a group membership protocol is agreement: all the membership sets of nonfaulty processors should be identical. Furthermore, all nonfaulty processors should be included in that common membership set. These properties are proved in Theorem 1. The progress property that all faulty processors are promptly diagnosed and removed from the common membership set is proved in Theorem 2, but much of the justification is already present in the invariant required to establish Theorem 1. A corollary is that the common membership set contains at most one faulty processor. In addition, faulty processors are able to diagnose themselves, and do so promptly; this is proved in Theorem 3. These three theorems correspond to the requirements stated in Section 2.

[4] There is a bug in the algorithm as just described: p should exclude itself (not b) when $\mathbf{ack}(p)$ AND NOT $\mathbf{ack}(b)$ and p was the previous broadcaster and sent a *false* **ack** in that broadcast. The bug manifests itself only when where there are exactly three processors in the membership, and its correction affects the proof of Theorem 3. The bug and its correction were pointed out by N. Shankar in an email message on 31 January 1998, and was independently discovered by Sadie Creese and Bill Roscoe of Oxford University using the FDR model checker.

Theorem 1 (Agreement). The local membership sets of all nonfaulty processors are always identical (and are called the *agreed set*) and contain all nonfaulty processors.

This theorem is proved by induction on **time**, but its statement must first be strengthened to yield an assertion that is inductive. In addition to claiming agreement among all nonfaulty processors, and that all nonfaulty processors are included in the agreed membership set, we must claim that all nonfaulty processors have the same values for the **ack** bits at each step, that these bits indeed reflect the intended meaning, and some additional facts about the diagnosis of earlier errors. These are needed to guarantee that in steps in which a fault has been detected, but not yet accurately ascribed, the next expected broadcaster will be nonfaulty and will resolve the uncertainty.

The invariant has the following conjuncts.

(1) All nonfaulty processors have the same local **membership** sets.
(2) All nonfaulty processors are in their own local **membership** sets.
(3) All nonfaulty processors have the same value for **ack**.
(4) For each processor p, $\mathbf{ack}(p)$ is true iff in the most recent previous step in which p expected a broadcast from a processor b, either p was b, or $arrived(b, p) \wedge (\mathbf{ack}(b) \vee \neg\mathbf{ack}(p))$ in that step.
(5) If a processor p became faulty less than n steps ago and q is a nonfaulty processor, either p is the present broadcaster or the present broadcaster is in p's local membership set iff it is in q's.
(6) If a receive fault occurred to processor p less than n steps ago, then either p is not the broadcaster or $\mathbf{ack}(p)$ is *false* while all nonfaulty q have $\mathbf{ack}(q) =$ *true*, or p is not in its local membership set.
(7) If in the previous step b is in the broadcaster slot, p is a nonfaulty processor, and $arrived(b, p)$ does not hold, then b is faulty in the current step.
(8) If the broadcaster b is expected by a nonfaulty processor, then b is either nonfaulty, or became faulty less than n steps ago.

Note that since all nonfaulty processors have identical membership sets and agree on which slot has been reached, they also agree on which processor is the next expected broadcaster. Moreover, by (5), processors that became faulty less than n slots ago agree with the nonfaulty ones on whether the present slot is expected to broadcast. The conjunct (5) is needed to show that newly faulty processors still agree with nonfaulty ones on the next expected broadcaster until they are diagnosed both by others and by themselves.

Conjuncts (7) and (8) are needed to guarantee that no fault has occurred to the processor in an expected slot following one that is newly silent. As mentioned earlier, the prompt diagnosis of receive faults seen in (6) is needed to prove (8). The fault arrival rate assumption thus links the seemingly independent questions of how soon a fault is followed by a (possibly ambiguous) indication that *some* fault has occurred, and how soon after that another fault can occur.

An important feature of the protocol is used in the proof and will be called the *restricted-change* lemma: if a change is made in the local membership set of p relative to the previous step, it is either in the membership of p itself, or in the membership of the broadcaster in the previous step. This can be seen easily in the description of the protocol. Another useful property that can be seen directly in the description of the protocol is that $\mathit{arrived}(b, p)$ will be *true* precisely when $b = p$ or (b is not actively send-faulty, b is in its own membership, p is not actively receive-faulty, and both b and p are in p's membership).

The conjuncts (4) and (7) simply record the intended meanings of **ack** bits and the non-receipt of a broadcast, and follow directly from the assignments to **ack** and the definition of *arrived*, respectively. We show the inductive argument for (5), (6), and (8) separately, and then return to (1), (2), and (3).

Lemma [for conjunct (5)]: If the invariant has been true so far, conjunct (5) will be true of the next step. That is, if in the next step p became faulty less than n steps ago, and q is nonfaulty, then either p is the broadcaster in that step, or the broadcaster is in p's local membership set iff it is in q's.

Proof: Let r denote the broadcaster in the next step. If $p = r$, the lemma holds. Otherwise, $n - 1$ steps ago r was in p's membership iff it was in q's, because both p and q were then nonfaulty and agreed on their membership sets. In all steps since then and up until the next step, r is not the broadcaster and is not p, and thus its membership in p's local membership set is not changed, by the restricted-change lemma. If $q \neq r$, the same reasoning holds for q and r, and we are done. If $q = r$, by the inductive hypothesis, n steps ago the local membership sets of q and p both contained q (when both were nonfaulty), and q still contains itself, since it is nonfaulty in all steps up to the next step, while q is in the membership set of p by reasoning as before. □

Lemma [for conjunct (6)]: If the invariant has been true so far, conjunct (6) will be true of the next step. That is, in the next step, if a receive fault occurred to processor p less than n time ago, then either p is not the broadcaster or $\mathbf{ack}(p)$ is *false* while all nonfaulty q have $\mathbf{ack}(q) = \mathit{true}$, or p is not in its local membership set.

Proof: If in the next step p did not become faulty less than n steps earlier, or p is not the broadcaster, the assertion is true. Otherwise, since p became faulty less than n steps earlier, and is now the broadcaster, it was not the broadcaster since it became faulty. Thus until the next step (inclusive) the broadcaster in each step was in p's local membership set if and only if it was in q's, for any nonfaulty q, by (5). If in all previous steps after p became receive-faulty, p and any nonfaulty q did not have the broadcaster at each step in their membership sets, then the **ack** bit of p is *false*. This is true because it was set to *false* in the step it became receive-faulty (by definition of **ack** and a receive fault) and has not been changed since (again, using the definition of **ack** for nonexpected slots). Similarly, for all nonfaulty q in this case, their **ack** bit is *true*: it was set to *true* when p became receive-faulty (since they all did receive a broadcast from a nonfaulty processor with which they had the same **ack** bit) and has not been changed since. Thus the assertion holds in this case.

If there was a step since p became receive-faulty, but earlier than the next step in which p and any nonfaulty q had the broadcaster r in their local membership sets, then that r must be nonfaulty: by the fault arrival assumption it is not a previously nonfaulty one that is newly send-faulty or receive-faulty within the last n time units (because p has become faulty within the last n), and by the inductive hypothesis, if it had become receive or send-faulty more than n units ago, it would already have been diagnosed in its previous broadcast slot or earlier and thus would not be expected. (Actually, by (6) in the step after its broadcast following its becoming faulty, it would not be in the local membership set of any nonfaulty process.)

So r is nonfaulty and thus will have $\mathbf{ack}(r)$ *true* in its broadcast. If p did not receive that broadcast, then it did not receive two consecutive expected broadcasts and thus removed itself by rule (a). If p did receive the broadcast, it removed itself by rule (b) because it received $\mathbf{ack}(r)$ as *true* while $\mathbf{ack}(p)$ was *false*. Thus in the present step, p is not in its own local membership, as required by the assertion. □

Lemma [for conjunct (8)]: If the invariant has been true so far, conjunct (8) is true in the next step. That is, if the broadcaster in that step p is expected by a nonfaulty processor q, then p is either nonfaulty or became faulty less than n steps ago.

Proof: By contradiction. Consider a situation where broadcaster p is expected by nonfaulty processors q, but p became faulty at least n steps earlier. Then there is an earlier step in which p is the broadcaster, and it became faulty less than n steps earlier. By conjunct (6), if it became receive-faulty, then it will not be in the membership set of any nonfaulty processor in the step following its broadcast (using the conditions for removing a broadcaster), contradicting the fact that it is expected now. If it became send-faulty, it also is not in the membership of any nonfaulty processor in the following step, by rule (c), again contradicting the hypothesis. □

Theorem 1 follows easily from the following claim.

Claim. The conjuncts (1)–(8) are an invariant.

The Proof is by induction.

Basis: All processors are nonfaulty initially and are in all local membership sets, the **ack** bits agree, and there have been no faults.

Inductive step: Conjuncts (5), (6), and (8) have already been proved, while (4) and (7) are simple inductions using the definitions of the terms. Here we show the remaining conjuncts (1)–(3): membership sets of nonfaulty processors agree, they contain the nonfaulty processors, and the **ack** bits of nonfaulty processors agree.

Assume the invariant is satisfied in all steps up to and including the m'th step (that can be identified with the value of **time** in the state). Consider the $m+1$'st. If the processor at slot $(m \bmod n)$ is not a member of the agreed set, nothing changes in step $m+1$ except the update of **time**, and the result follows.

Otherwise, the processor at slot $(m \bmod n)$ is in the agreed set and is expected by nonfaulty processors. If it is nonfaulty, it will broadcast, be received by all nonfaulty processors, and be maintained in their local membership set (the broadcast and local **ack** bits agree by the inductive hypothesis). It also retains itself in its local membership set. All nonfaulty processors will set **ack** to *true* in the next step. No nonfaulty processor will remove itself. This is true because: condition (a) does not hold, since a nonfaulty processor will broadcast and the message is expected and thus is received; condition (b) does not hold because the agreed sets were the same in all previous stages, as were the **ack** bits. Thus all nonfaulty processors still have the same local membership sets and **ack** bits, and include themselves in their local membership sets.

If the processor b at slot $(m \bmod n)$ is in the agreed set but has a send fault or has detected its own receive fault and removed itself from its local membership set, no nonfaulty processor p will receive b even though it was expected (i.e., $arrived(b, p)$ is *false*), and all will mark it as absent in step $m+1$ by rule (c) and will set **ack** to *false* in that step. No nonfaulty processors will remove themselves in this case: since $arrived(b, p)$ is *false*, condition (b) is irrelevant, and (a) also does not hold, because the neighbors in expected slots around the silent processor must be nonfaulty, by the fault model and the conjuncts (7) and (8). In particular, the broadcast in the most recent expected slot before b was from a nonfaulty processor and thus must have arrived at p and had an **ack** bit that agreed with that of all nonfaulty recipients (by the inductive hypothesis). Therefore the **ack**(p) bit in step m is *true* by conjunct (4). Thus in step $m+1$ the local membership sets and the **ack** bits of those nonfaulty processors remain identical, and no nonfaulty processor removes itself.

If the processor b in the broadcast slot is in the agreed set and in its local membership set (and thus is expected by nonfaulty processors) but is receive-faulty, then by conjunct (8) the receive fault occurred within the last n steps, and by conjunct (6), b will broadcast **ack**(b) as *false*, while nonfaulty processors p have **ack**$(p) = true$. Thus when the new broadcast occurs, all nonfaulty processors will remove the receive-faulty broadcaster by rule (d), and also set **ack** to *false*. In this case too, no nonfaulty processor will remove itself from its local membership set: since $arrived(b, p)$ is *true*, condition (a) is irrelevant, and condition (b) does not hold since the broadcaster had a *false* **ack** bit when it broadcast. □

Most of the justification for prompt diagnosis and removal of faulty processors was provided in the proof of the invariant above. We have:

Theorem 2 (Prompt Removal). A faulty processor is removed from the membership sets of nonfaulty processors in the step following its first broadcast slot while faulty.

Proof: As proved in conjunct (6) of the invariant, if a processor p becomes receive-faulty, then in its next broadcast either **ack**(p) is *false*, while **ack**(q) is *true* for nonfaulty processors q, or p is not in its local membership set. In the former case, p will be removed from the local membership set of q by rule (d) and in the latter case $arrived(p, q)$ is *false* so that p is removed by rule (c). If p becomes send-faulty, again $arrived(p, q)$ is *false*, so p is removed by rule (c). □

Corollary (One Faulty Member). In any step the agreed group contains at most one faulty processor.

Proof: Immediate from Theorem 2 and the fault arrival rate assumption. □

As part of the proof of the invariant needed for Theorem 1, in conjunct(6), we showed that a processor that is not the next expected broadcaster after becoming receive-faulty will remove itself from its local membership set. Here we show that any faulty processor, including send-faulty ones and those that became receive-faulty just before broadcasting, will remove themselves from their local membership sets.

Theorem 3 (Rapid Self-Diagnosis). A newly faulty processor will remove itself from its local membership set (and thereby diagnose itself) when the slots of at most two nonfaulty processors have been passed.

Proof: If a processor p becomes send-faulty, all nonfaulty processors will set their **ack** bits to *false* in the step following that processor's slot, since the slot is expected and no message is received. Similarly, if p just became receive-faulty in the expected broadcast before its slot, it will broadcast **ack** as *false*, while the nonfaulty processors have **ack** $=$ *true*, and thus will set their **ack** bits to *false*. In either case, p will set its **ack** bit to *true* in the step after it broadcasts. Until its own or the previous broadcast, p was nonfaulty, and thus its local membership set agreed with all other nonfaulty processors. By the invariant of Theorem 1, no nonfaulty processor will remove itself due to the new fault, thus the next expected slot of the nonfaulty processors is the same as the next expected slot of the faulty one. By the fault arrival assumption, the next expected slot must be nonfaulty, since it cannot be newly faulty, and by the invariant it cannot be an undiagnosed old fault. Thus the newly faulty processor p will receive **ack** $=$ *false* in the message from the next expected slot, disagree with the broadcaster, and set its own **ack** bit to *false*. Since all nonfaulty processors receive that broadcast and agree with its **ack** bit, p will receive **ack** as *true* in the expected slot after that (using the fact that there are at least two nonfaulty processors within the group). At that point, the faulty processor p will remove itself from its local membership set, using (b).[5]

If a processor p becomes receive-faulty in its transition to the next step, but p is not the next expected broadcaster, it will remove the broadcaster from its local membership set, but otherwise has the same local membership sets and next expected slot as the nonfaulty ones. It will also set $\mathbf{ack}(p)$ to *false*. Again by the fault model, the next expected broadcaster must be nonfaulty, will broadcast **ack** as *true*, and the receive-faulty processor p will remove itself using (b). $\square$

5 Discussion and Conclusions

The fault arrival rate we assume in our fault model is at most one new faulty processor in any consecutive $n+1$ slots. This is clearly tight, since if n were used in place of $n+1$, the algorithm fails. Consider a scenario with a receive fault of the processor just before the broadcaster, followed n steps later by a send fault of that same broadcaster. Since the receive-faulty processor will self-diagnose and fall silent in its slot just before the subsequent send fault, all nonfaulty processors will not receive two consecutive expected broadcasts. They will all then incorrectly remove themselves from their local membership sets.

As this analysis shows, clustered faults can cause our algorithm to fail, though it is generally more robust than this worst case analysis suggests: the precise requirements are that the expected broadcaster must be nonfaulty when it follows a receive fault (unless it was that broadcaster that suffered the receive fault in the previous step) or the silence perpetrated by a successfully self-diagnosed receive fault, and the next *two* expected broadcasters must be nonfaulty following a send fault or the broadcast of a processor that suffered a receive fault in the previous step. Stochastic measurements are needed to determine more representative measures of the fault arrival rates and patterns that can be tolerated.

The requirement of two nonfaulty processors is also tight: if there are two processors remaining in the group and one of them becomes faulty, then there is no longer any possibility of distinguishing between a send and a receive fault. In either case, a broadcast is not received by the other processor, each will ultimately remove the other from its local membership set, and neither will ever self-diagnose.

5.1 Robustness of Assumptions

Self-diagnosis requires a very strong assumption on the behavior of faulty processors: namely, that they continue to execute the algorithm correctly for upto n steps after becoming faulty. This is plausible when a send or receive fault is truly due to a communications problem, but is less so when it is the manifestation of a more serious failure. As noted previously, we assume that our group

[5] As noted in footnote 4, this argument is incorrect when there are only three processors. In the corrected algorithm, p excludes itself on the previous round, when it receives $\mathbf{ack} = \mathit{false}$.

membership protocol is part of a larger suite of protocols that can mask other types of faults, or transform their manifestations into those that can be tolerated by our algorithm. For example, checksums transform corrupted communications into the send and receive faults that our algorithm can tolerate.

Loss of clock synchronization or a "babbling idiot" failure could cause a processor to transmit outside its own slot, thereby possibly preventing nonfaulty processors from communicating on the bus and leading to complete failure of the system. These failure modes are handled using "Bus Guardians" in TTP and self-checking pairs in ARINC 659: in both cases, each processor's access to the bus is mediated by a second component with an independent clock, so that coincident double faults are required to create a catastrophic failure. Computation faults are likewise detected by pairwise comparison, or masked by voting, and higher-level diagnosis and recovery algorithms then isolate or reboot the afflicted processor [15, 18–20]. In these cases, the failure mode is reduced to fail-silence, which manifests itself to our group membership protocol as a combination of long-duration send and receive faults for the processor concerned. Since all nonfaulty processors promptly detect and exclude send-faulty processors without requiring the processor concerned to diagnose itself, the safety properties of group membership are preserved even if the faulty processor is unable to execute the protocol correctly. Self-diagnosis of send and receive faults, which is assured for processors that do correctly execute our protocol, is moot for processors that are in the grip of some larger crisis.

Processor faults that are not masked or detected and reduced to fail-silence by other components of the protocol suite can be catastrophic for group membership if they cause a processor to violate the protocol. For example, if the next expected processor following a send fault incorrectly sends **ack** = *true*, then all other processors will diagnose themselves as receive-faulty and exclude themselves from the group. In TTP, such a collapse causes reversion to the blackout operating mode. More insidious execution faults could allow a processor that suffers a receive fault not to diagnose its failure. Since receive faults (other than of the next expected broadcaster) require the afflicted processor to signal its failure (by remaining silent), such an execution fault would allow the faulty processor to remain a member of the group.

The possibility of such catastrophic or insidious faults is the price paid for the low overhead of our group membership protocol. Experiments with an architecture similar to TTP [6] show that the incidence of such fail-silence violations is sufficiently rare that they present an acceptably low risk to the system (given the other mechanisms present in the total suite of protocols).

5.2 Formal Analysis, and Future Work

We have described and proved correct a protocol for synchronous group membership that, driven by practical considerations, trades a very restrictive fault model in return for very low communications overhead—just one bit per message. Despite the paucity of information carried by each message, the protocol allows rapid and accurate identification and elimination of faulty processors.

The reasoning that supports this claim, however, requires inferences to be made over *sequences* of messages; this, in turn, requires the statement of correctness to be strengthened significantly in order to obtain one that is inductive and requires a surprisingly intricate proof with extensive case-analysis. We found determination of an adequately strong (and true!) invariant, and development of the corresponding proof, to be quite challenging, and turned to formal techniques for assistance. We used the Murϕ state-exploration system [3] to examine instances of the protocol for the purposes of debugging the protocol, its fault model and its assumptions, and also to check candidate invariants. Using Murϕ, we were able to exhaustively check the behaviors of a ring of six processors with up to three faults. This required some tens of minutes (on a Sparc 20) and 100 MB of memory and entailed exploration of almost two million states. We are currently formalizing the general case and subjecting our proof of correctness to mechanical checking using the PVS verification system [13].

For the future, we are interested in systematic techniques for deriving strengthened invariants of the kind needed here, and for generating the proof of correctness. Some of the reasoning resembles that seen in the backward reasoning of precedence properties in temporal logic [11].

The group membership protocol presented here has no provision for readmitting previously-faulty processors that now appear to be working correctly again. Simple extensions, such as allowing a repaired processor to just "speak up" when its slot comes by, are inadequate. (A processor that has a receive fault just as the new member speaks up will not be aware of the fact and its local membership set will diverge from that of the other processors; a second fault can then provoke catastrophic failure of the entire system.) We are aware of solutions that do work, at the cost of strong assumptions on the fault-detection capability of the CRCs appended to each message, and plan to subject these to formal examination. TTP encodes its "critical state" in its CRC calculation, and the ack bit of our abstract protocol is in fact encoded implicitly in the CRC and recovered by recalculation of the CRC for each of the two possible values represented by that bit.

We are also eager to explore more of the highly optimized and integrated algorithms seen in industrial protocols for safety-critical distributed systems, such as TTP and ARINC 659. For example, the restrictive fault model used for our group membership protocol is partly justified by the existence of a blackout operating mode to deal with more severe, or clustered, faults. An interesting challenge for the future is to establish the fault coverage of this combination, and the correctness of the transitions between different operating modes in the presence of faults.

References

Papers by SRI authors are generally available from `http://www.csl.sri.com/fm.html`.

1. *ARINC Specification 659: Backplane Data Bus.* Aeronautical Radio, Inc, Annapolis, MD, December 1993. Prepared by the Airlines Electronic Engineering Committee.
2. Flaviu Cristian. Reaching agreement on processor-group membership in synchronous distributed systems. *Distributed Systems*, 4:175–187, 1991.
3. David L. Dill. The Murϕ verification system. In Rajeev Alur and Thomas A. Henzinger, editors, *Computer-Aided Verification, CAV '96*, volume 1102 of *Lecture Notes in Computer Science*, pages 390–393, New Brunswick, NJ, July/August 1996. Springer-Verlag.
4. 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.
5. *Fault Tolerant Computing Symposium 25: Highlights from 25 Years*, Pasadena, CA, June 1995. IEEE Computer Society.
6. Johan Karlsson, Peter Folkesson, Jean Arlat, Yves Crouzet, and Güther Leber. Integration and comparison of three physical fault injection techniques. In Brian Randell, Jean-Claude Laprie, Hermann Kopetz, and Bev Littlewood, editors, *Predictably Dependable Computing Systems*, Basic Research Series, pages 309–327. Springer, 1995.
7. H. Kopetz. Automotive electronics—present state and future prospects. In *Fault Tolerant Computing Symposium 25: Special Issue*, pages 66–75, Pasadena, CA, June 1995. IEEE Computer Society.
8. H. Kopetz, G. Grünsteidl, and J. Reisinger. Fault-tolerant membership service in a synchronous distributed real-time system. In A. Avižienis and J. C. Laprie, editors, *Dependable Computing for Critical Applications*, volume 4 of *Dependable Computing and Fault-Tolerant Systems*, pages 411–429, Santa Barbara, CA, August 1989. Springer-Verlag, Vienna, Austria.
9. Hermann Kopetz and Günter Grünsteidl. TTP—a protocol for fault-tolerant real-time systems. *IEEE Computer*, 27(1):14–23, January 1994.
10. 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. Reprinted in [5, pp. 438–447].
11. Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety.* Springer-Verlag, 1995.
12. Fred J. Meyer and Dhiraj K. Pradhan. Consensus with dual failure modes. *IEEE Transactions on Parallel and Distributed Systems*, 2(2):214–222, April 1991.
13. Sam Owre, John Rushby, Natarajan Shankar, and Friedrich von Henke. Formal verification for fault-tolerant architectures: Prolegomena to the design of PVS. *IEEE Transactions on Software Engineering*, 21(2):107–125, February 1995.
14. John Rushby. A formally verified algorithm for clock synchronization under a hybrid fault model. In *Thirteenth ACM Symposium on Principles of Distributed*

Computing, pages 304–313, Los Angeles, CA, August 1994. Association for Computing Machinery.
15. John Rushby. Reconfiguration and transient recovery in state-machine architectures. In *Fault Tolerant Computing Symposium 26*, pages 6–15, Sendai, Japan, June 1996. IEEE Computer Society.
16. Fred B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys*, 22(4):299–319, December 1990.
17. 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.
18. C. J. Walter, N. Suri, and M. M. Hugue. Continual on-line diagnosis of hybrid faults. In F. Cristian, G. Le Lann, and T. Lunt, editors, *Dependable Computing for Critical Applications—4*, volume 9 of *Dependable Computing and Fault-Tolerant Systems*, pages 233–249. Springer-Verlag, Vienna, Austria, January 1994.
19. Chris J. Walter. Identifying the cause of detected errors. In *Fault Tolerant Computing Symposium 20*, pages 48–55, Newcastle upon Tyne, UK, June 1990. IEEE Computer Society.
20. Chris J. Walter, Patrick Lincoln, and Neeraj Suri. Formally verified on-line diagnosis. *IEEE Transactions on Software Engineering*, 23(11):684–721, November 1997.

This article was processed using the LaTeX macro package with LLNCS style