# The Formal Verification of a Reintegration Protocol

Lee Pike
Formal Methods Group
NASA Langley Research Center
lepike@indiana.edu

Steven D. Johnson
Department of Computer Science
Indiana University, Bloomington
sjohnson@cs.indiana.edu

## ABSTRACT

We report the first formal verification of a reintegration protocol for a safety-critical distributed embedded system. A *reintegration protocol* increases system survivability by allowing a transiently-faulty node to regain state. The protocol is verified in the Symbolic Analysis Laboratory (SAL), where bounded model-checking and decision procedures are used to verify infinite-state systems by *k-induction*. The protocol and its environment are modeled using a recently-developed explicit real-time model. Because $k$-induction has exponential complexity, we optimize this model to reduce the size of $k$ necessary for the verification and to make $k$ invariant to the number of nodes. A corollary of the verification is that a *clique avoidance* property is satisfied.

**Categories and Subject Descriptors:** D.2.4 [Software Engineering]: Software/Program Verification

**General Terms:** Verification

**Keywords:** reintegration protocol, infinite-state systems, real-time, formal verification, infinite-state bounded model-checking

## 1. INTRODUCTION

Digital control systems (i.e., "x-by-wire") are now being designed for use in safety-critical environments such as automobiles, commercial aircraft, and piloted space vehicles. In a single vehicle, many systems require reliable real-time intercommunication. Highly-reliable fault-tolerant virtual buses are being designed for this purpose, including the Time-Triggered Architecture, SAFEbus, FlexRay, and NASA Langley's SPIDER [13].

These buses (Rushby notes that the term 'bus' "understates their complexity, sophistication, and criticality" [13]) are themselves implemented as synchronized distributed systems. A node in the distributed system may suffer a *transient fault* causing it to lose its volatile state but suffer no permanent damage. Although such a node may be fault-free, its state no longer is coordinated with that of the *operational clique*, the set of non-faulty nodes with coordinating states that provide the system services.[1] For a transiently-faulty node to regain correct state, it executes a *reintegration protocol*. Its main purpose is to allow the reintegrating node (called the *reintegrator*) to resynchronize its local clock with those of the nodes in the operational clique. This is a necessary precondition for it to regain other state (e.g., the dynamic schedule).

We present a formal verification of the SPIDER Reintegration Protocol [15]. This is the first formal verification of a reintegration protocol. Pfeifer and Rushby each have stated that the formal analysis of reintegration remains important future work for TTA, one of the most mature and fully formally-verified fault-tolerant buses in development [9, 14]. This work should be immediately extensible to the formal verification of reintegration protocols for other fault-tolerant systems.

The approach builds on recent developments in real-time system verification involving *explicit* real-time models in which the current time is tracked with a variable, and discrete events have real-time bounds on when they can occur [5, 7] (in [7], only explicit discrete-time models are considered). This contrasts with *implicit* (or clock-based) real-time models, such as timed automata [1]. The attraction of explicit real-time models is that their syntax is simple (time constraints are essentially inqualities over the reals), and they require no special semantics for verification. Furthermore, parameterized specifications can be verified. For example, in this verification, the maximum clock skew $\pi$ is modeled as an uninterpreted constant, so the verification holds for any value of $\pi$.

This work extends results in using bounded model-checking and decision procedures to verify infinite-state real-time systems using $k$-induction, a generalization of the ordinary induction principle over transition systems. This is done in the generic (i.e., there are no special provisions for modeling real-time systems) model-checker SAL [4]. Because the technique is exponential with respect to $k$ (where $k$ is the length of trajectories over which induction is taken), a focus of this work is to reduce the size of $k$ and make it invariant to the number of nodes modeled. Our two approaches are to optimize a recently-developed explicit real-timed model called *timeout automata* [5] and to provide a time-triggered model of event-triggered systems.

[1]The operational clique is not necessarily equivalent to the set of non-faulty nodes: for example, a reintegrating node is a non-faulty node outside the operational clique. This distinction can be subtle and is in fact responsible for an error in the previous design of another SPIDER protocol [11].

## 2. EXPLICIT REAL-TIME VERIFICATION

First, we describe the SAL toolset used. Next, we describe the formal timing model used in the verification (for space considerations, the model is described informally; see [10] for more details).

### 2.1 SAL

The protocol is specified and verified in SAL, developed by SRI, International [4]. SAL is a toolset that includes explicit-state, symbolic, and bounded model-checkers, an interactive simulator, as well as other tools. A single language serves as the input to the verification tools. The fundamental building block in the language is a *module* specifying a state machine. Modules can be synchronously or asynchronously composed, and they communicate via shared variables. SAL has provisions for both finite and infinite types, uninterpreted and interpreted constants and functions, and quantification over finite domains.

The verification tools used were SAL's bounded model-checker in conjunction with the Integrated Canonizer and Solver (ICS), a decision procedure for a quantifier-free, first-order theory of equality, the terms of which include uniterpreted functions, linear arithmetic, products, arrays, fixed-sized vectors, etc. [4].[2] Together, these tools can be used to prove state invariants hold in infinite transition systems. The invariants do not need to be strictly inductive; SAL supports *k-induction*, a generalization of the ordinary induction principle (over transition systems) [3].

*Definition 1.* For a transition system with a set of states $S$, a set of initial states $S^0 \subseteq S$, and a binary transition relation $\rightarrow$, a property $P$ is **k-inductive** if for each trajectory $s_0 \rightarrow s_1 \rightarrow \ldots \rightarrow s_k$ of length $k$, if $s_0 \in S^0$, then $P$ holds at every state on the trajectory (the base case), and if $P$ holds at states $s_0$ through $s_{k-1}$, then $P$ holds in $s_k$ (the induction step).

The ordinary induction principle is the special case when $k = 1$. The benefit of $k$-induction is that, as $k$ increases, weaker invariants may be provable. Furthermore, SAL allows previously-proved invariants to be used as lemmas in $k$-induction proofs. A lemma strengthens the antecedents in the base case and induction step so that only states satisfying the lemma are considered.

### 2.2 Timeout Automata

Dutertre and Sorea develop an explicit real-time model called *timeout automata* for $k$-induction verification of infinite-state real-time systems in SAL. Timeout automata were motived by the model of execution used in discrete-event simulation [2]. The model includes a set of state variables (which potentially range over infinite sets) and a set of *timeout* variables ranging over the reals. These denote when some portion of the state is to update. Additionally, there is a *clock* variable ranging over the reals that denotes the current time. In the transition system constructed, there are two kinds of transitions. In *discrete transitions*, some portion of the state variables update. These transitions are enabled if the clock value is equal to the appropriate timeout In each of these transitions, the timeout is (usually nondeterministically) updated to a new value strictly greater than its current value. In *time transitions*, the clock is updated to the minimum of the timeouts.

[2]It is possible to use other decision procedures with SAL.

Whereas in timed automata clocks measure how much time has elapsed since their last reset, in timeout automata, timeouts measure how much time will elapse until the next discrete transition. Very loosely speaking, timeout automata and timed automata are dual with respect to their perspective of time.

Proofs by $k$-induction have a complexity that is exponential with respect to $k$ (by solving the equivalent boolean satisfaction problem). The initial timeout automata models of the SPIDER Reintegration Protocol required $k$-induction at infeasible depths. We therefore optimize the model in two ways to reduce size of $k$ required to prove by $k$-induction.

1. The semantics of timeout automata (and other real-time models) include time and discrete transitions. However, the only purpose of the clock variable is to record the current time. This can be implicitly determined by the least-valued timeout. Thus, we can remove this variable as well as the corresponding time transitions. Then the model contains only discrete transitions, and a discrete transition is enabled just in case the corresponding timeout is the minimum of the timeouts.

2. Communication between subsystems (each specified by a module) is modeled with shared variables SAL. In the original timeout automata semantics, the modules are asynchronously composed, and two sequential discrete transitions are taken: a sender updates a shared variable, then the receiver updates its state based on the value of the variable. This can be reduced to a single transition by synchronously composing the modules. In a synchronous composition in SAL, the transition a module can be a function of the next-state values for the state variables of another module. Thus, the receiver updates its state concurrently with the sender writing the shared variable.

These optimizations reduce the state space and significantly reduce the size of $k$ required in a proof. For example, verifying by $k$-induction a safety property for the ubiquitous train-gate-controller real-time system can be reduced from $k = 14$ to $k = 5$ when these optimizations are made [10].

## 3. THE REINTEGRATION PROTOCOL

In the following, we describe both the reintegration protocol and the system assumptions that must hold for it to execute correctly.

During the reintegration protocol, the reintegrator monitors its communication links for *echos*. Echos are messages sent by nodes during the *SPIDER Clock Synchronization Protocol*, a formally-verified fault-tolerant protocol in which operational nodes periodically synchronize their local clocks [8]. The period from one execution of the protocol to the next is called a *resynchronization frame*. The correctness of the reintegration protocol depends on the following two properties holding:

1. The *frame property* ensures that frames are long enough for the reintegration protocol to behave correctly, and is a function of the reintegrator's accusations. If the reintegrator observes faulty behavior from a monitored node, it *accuses* it, and then ignores its subsequent messages. The frame property is that $P > l\pi + 2\pi$,

where $P$ is the length of each frame, $l$ is the number of unaccused faulty nodes, and $\pi$ is the maximum clock skew between operational nodes.

2. The *Majority Property* is that the majority of the nodes that have not been accused by the reintegrator during the protocol are operational nodes. The property ensures that enough of the unaccused monitored nodes are non-faulty for the protocol to work. It serves as the *maximum fault assumption* for this protocol.[3]

The reintegration protocol is given in Fig. 1. It is comprised of three modes of operation: *preliminary diagnosis*, *frame synchronization*, and *synchronization capture*. These modes are executed sequentially as shown in Fig. 1.

The following state variables determine the state of the reintegrator during the execution of the protocol. Let $i$ range over the indices of the nodes the reintegrator monitors. Let *accs* and *seen* be arrays of booleans and integers, respectively, where *accs*[$i$] is true if the reintegrator *accuses* node $i$, and *seen*[$i$] is a counter marking how many echo messages the reintegrator has observed from $i$. Real-valued variables include *clock*, *fs_finish*, and *pd_finish*, such that *clock* is the current value of the reintegrator's local clock, *fs_finish* is a timer for the frame synchronization mode, and *pd_finish* denotes the time at which the preliminary diagnosis mode completes. State variables are initialized as shown in Fig. 1. The predicate "`when` *echo*($i$)" holds at the moment the reintegrator receives an echo message from node $i$.

The purpose of preliminary diagnosis is to acquire diagnostic data to recognize faulty nodes early in the protocol by monitoring echo messages for the duration $P + \pi$. The purpose of the frame synchronization mode is to determine a time at which all operational nodes have already issued an echo message in some frame and before any operational node issues an echo in the next frame. Frame synchronization provides the reintegrator with a coarse-grained synchronization with the operational clique. The purpose of the synchronization capture mode is to allow the reintegrator to synchronize with an echo from some operational node. It does so by synchronizing when it has received echos from at least half of the nodes it has not accused (or has not already seen in this mode).

```
for each i, accs[i] := false;
for each i, seen[i] := 0;
mode := prelim_diag;

pd_finish := clock + P + π;
while clock < pd_finish do {
  for each i, when echo(i) do {
    if (seen[i] < 2 and not accs[i])
    then seen[i] := seen[i] + 1
    else accs[i] := true}};
for each i, if seen[i] = 0 then accs[i];
mode := frame_synch;

for each i, seen[i] := 0;
fs_finish := clock;
while clock − fs_finish < π do {
  for each i, when echo(i) do {
    if (seen[i] = 0 and not accs[i])
    then {fs_finish := clock;
          seen[i] := seen[i] + 1};
    else accs[i] := true}};
mode := synch_capture;

for each i, seen[i] := 0;
while     |{i | seen[i] > 0}|
       ≤ |{i | not accs[i]}|/2 do {
  for each i, when echo(i) do {
    if (seen[i] = 0 and not accs[i])
    then seen[i] := seen[i] + 1}};
clock := 0;
```

**Figure 1: The Reintegration Protocol**

## 4. TIME-TRIGGERED MODELING

One particular challenge is modeling faulty nodes so that arbitrary behaviors are possible, while ensuring $k$-induction proofs are feasible. Because the reintegrator is unsynchronized and event-triggered, in a naïve model of the entire system, the reintegrator would make a transition whenever it receives an echo from a monitored node. In a formal model, this amounts to updating its timeout to the time at which the next echo message is observed and updating its state accordingly, for every successive echo message. However, because a faulty node may issue multiple echos before being ignored by the reintegrator, this model can quickly lead the reintegrator to make a large number of state transitions. For $k$-induction to succeed, a more sophisticated model is required.

A preferable model is one in which the reintegrator's transitions are essentially time-triggered. This amounts to the reintegrator updating its timeout at regular intervals and updating its state based on all of the echos received during each interval. Care must be taken to make a time-triggered model of event-triggered behavior conservative. In a time-triggered timeout update, the reintegrator "observes" those echos that come after its current timeout and no later than the time at which it sets its next timeout. For example, suppose the reintegrator were to update its state in a time-triggered fashion as illustrated in Fig. 2, where `reint_to` is the reintegrator's current timeout, and `reint_to'` is its updated timeout. During this transition, it may observe *echo*, but then the node issuing *echo* may issue another message, *echo'*, that is unobserved by the reintegrator.

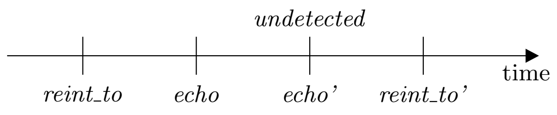

**Figure 2: Missed Echo Messages**

Therefore, faulty nodes issue multiple echo messages in a single transition. The model of faulty nodes must allow them the possibility of non-faulty behavior. The echos are nondeterministically updated so that they may be issued arbitrarily far in the future. All of the echos a node issues that are beyond the reintegrator's updated timeout are ignored. This models the full range of possible faulty behaviors.

[3]Any proof of fault-tolerance requires an assumption about the kinds of and maximum number of faults present.

## 5. VERIFYING THE PROTOCOL

There are two main theorems to prove: that the reintegrator accuses no operational nodes during the execution of the protocol, and that the reintegrator successfully resynchronizes with the operational nodes upon completion of the protocol.

THEOREM 1 (NO OPERATIONAL ACCUSATIONS). *For all operational nodes $i$, $accs[i]$ does not hold during the reintegration protocol.*

THEOREM 2 (SYNCHRONIZATION ACQUISITION). *For all operational nodes $i$, $|clock - echo(i)| < \pi$ upon termination of the reintegration protocol.*

The proofs of these theorems via $k$-induction requires a number of supporting lemmas. Our strategy is to decompose the verification into a verification of its constituent modes. Each mode should guarantee certain postconditions. The postconditions for a mode then serve as preconditions for succeeding modes. This strategy can be followed through the entire protocol making the proof of the above theorems straightforward. This proof strategy is inspired by the proof-by-abstraction techniques used in [5, 6].

For all but a few minor lemmas, the time-triggered modeling approach makes the size of $k$ required invariant to the number of monitored nodes modeled. Rather, $k$ depends on the duration of a mode (i.e., for how many resynchronization frames it is active) rather than on how many echos are received in the mode. All lemmas are proved by $k$-induction for $k \leq 4$ for the verifications attempted.

The protocol has been verified for up to four monitored nodes (no more than one of which can be faulty to satisfy the Majority Property). The proofs took on the order of minutes to complete on typical modern desktop CPU. Strengthening the invariants would allow larger architectures to be verified.

*Clique avoidance* is the property that there exists exactly one operational clique in the system [13]. If more than one clique exists, the nodes in one clique will consider the nodes in the other to be either faulty or recovering. SPIDER and similar systems must satisfy clique avoidance. Theorem 1 is a necessary condition for clique avoidance. It ensures that if a clique exists, a non-faulty reintegrator cannot "ignore" the clique. Barring a massive correlated failure of the nodes, a second clique cannot be formed.

## 6. CONCLUSION

We have described a formal proof of a explicit real-time model of the SPIDER Reintegration Protocol in the SAL tool using $k$-induction. We have described improvements to a novel explicit real-time formalism for infinite-state bounded model-checking that has been successfully used now in two industrial-scale verifications (including this work and that presented in [5]). Furthermore, we have described a means by which event-triggered behavior can be modeled as time-triggered behavior.

The full SAL model, including a script to run the associated proofs, and an extended technical report [10] describing the modeling and verification can be found on-line at `http://shemesh.larc.nasa.gov/fm/spider/reint_sal/`.

The formal specification and verification of the reintegration protocol did not reveal any flaws in the protocol. Nevertheless, it was of value: No hand proofs existed to demonstrate its correctness and previous versions had flaws, the protocol is significantly different from the other SPIDER protocols and most published distributed protocols, and the formal verification strongly suggests that clique avoidance is preserved. The assumptions used in the formal verification generalize those stated in the design document [15] to a small extent.

## 7. ACKNOWLEDGMENTS

The authors thank Wilfredo Torres-Pomales of NASA Langley for describing the reintegration protocol in detail and repeatedly. Paul Miner, Jeffrey Maddalon, and Mahyar Malekpour made helpful suggestions. Bruno Dutertre and Leonardo de Moura provided helpful details concerning $k$-induction in SAL.

## 8. REFERENCES

[1] R. Alur. Timed automata. In *CAV*, volume 1633 of *LNCS*, pages 8–22, 1999.

[2] J. Banks and J. S. C. II. *Discrete-Event Simulation.* Prentice-Hall, 1984.

[3] L. de Moura et al. Bounded model checking and induction: From refutation to verification. In *CAV*, volume 2725 of *LNCS*, 2003.

[4] L. de Moura et al. SAL 2. In *CAV*, volume 3114 of *LNCS*, pages 496–500, 2004.

[5] B. Dutertre and M. Sorea. Modeling and verification of a fault-tolerant real-time startup protocol using calendar automata. In *FTRTFT*, volume 3253 of *LNCS*, pages 199–214, Sept. 2004.

[6] B. Dutertre and M. Sorea. Timed systems in SAL. Technical Report SRI-SDL-04-03, SRI, International, 2004.

[7] L. Lamport. Real time is really simple. Technical Report MSR-TR-2005-30, Microsoft Research, 2005.

[8] P. Miner, A. Geser, L. Pike, and J. Maddalon. A unified fault-tolerance protocol. In *FORMATS-FTRTFT*, volume 3253 of *LNCS*, pages 167–182, 2004.

[9] H. Pfeifer. *Formal Analysis of Fault-Tolerant Algorithms in the Time-Triggered Architecture.* PhD thesis, Universität Ulm, 2003.

[10] L. Pike. Real-time system verification by $k$-induction. Technical Report TM-2005-213751, NASA Langley Research Center, 2005.

[11] L. Pike, P. Miner, and W. Torres. Model checking failed conjectures in theorem proving: a case study. Technical Report NASA/TM–2004–213278, NASA Langley Research Center, November 2004.

[12] J. Rushby. Verification diagrams revisited: Disjunctive invariants for easy verification. In *CAV*, volume 1855 of *LNCS*, pages 508–520, 2000.

[13] J. Rushby. Bus architectures for safety-critical embedded systems. In *EMSOFT 2001*, volume 2211 of *LNCS*, pages 306–323, 2001.

[14] J. Rushby. An overview of formal verification for the time-triggered architecture. In *FTRTFT*, volume 2469 of *LNCS*, pages 83–105, 2002.

[15] W. Torres-Pomales, M. R. Malekpour, and P. Miner. ROBUS-2: A fault-tolerant broadcast communication system. Technical Report NASA/TM-2005-213540, NASA Langley Research Center, 2005.