

Modular Model-Checking of a Byzantine Fault-Tolerant Protocol

Benjamin F Jones¹ and Lee Pike¹

¹Galois, Inc. – <http://galois.com>

Challenge

Model checking distributed, fault-tolerant systems is challenging:

- ▶ large state spaces
- ▶ non-determinism
- ▶ complexity inherent in modeling multiple nodes behavior and faults

Coping with complexity

Model checking these systems has been accomplished by resorting to ad-hoc abstractions:

- ▶ local node behavior is abstracted (e.g. replaced by pre/post conditions)
- ▶ message passing is simplified using shared state
- ▶ systems are modeled at small, fixed scales

Ad-hoc abstractions tend to cause models to *diverge* from implementations, reducing their applicability.

Modularity

TODO rewrite

- ▶ Model checking these systems is further complicated when system behavior and fault behavior are mixed.
- ▶ Similarly, when trying to prove that a model satisfies a property, it is very useful to be able to reason about the system behavior and the fault behavior independently and then compose the proofs. The same goes for local node behavior versus and communication model.

What We Want in a Model

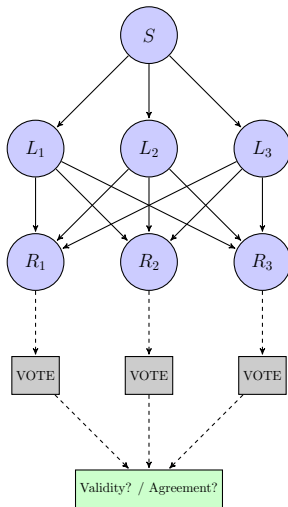
The premise for this work is the following ideal:

1. Implementation-level detail
2. Scalability of model checking
3. Modularity

What We've Done

- ▶ **Framework:** We've attempted to come up with a fairly general framework for modeling distributed, fault-tolerant systems in a way that gets us closer to the ideal of the 3 last points.
- ▶ **Case Study:** We've taken this framework and used it to model variations on the *Hybrid Oral Messages* algorithm that vary in their timing model, node behavior, and fault model.

Oral Messages with 1 Round



Hybrid Fault Model

In [1], T and Park introduced their “Hybrid Fault Model” which captures 4 different fault modes and a system for weighting their impact.

- ▶ Non-faulty
- ▶ Manifestly faulty
- ▶ Symmetrically faulty
- ▶ Byzantine faulty

OM(1) was shown to be tolerant of a certain weighted sum of these fault types. In the classical case where only byzantine faults are considered and there are 3 lieutenants, the system tolerates at most 1 fault.

Hybrid Oral Messages Algorithm

In the course of formalizing this result on fault-tolerance, it was discovered [2] that the proof was wrong. However, the OM(1) algorithm can be adapted to one that satisfies the conclusion of [1].

This algorithm, due to Lincoln and Rushby, is called OMH(1).

Three Systematic Abstractions

In this framework for model systems, we use three principal abstractions:

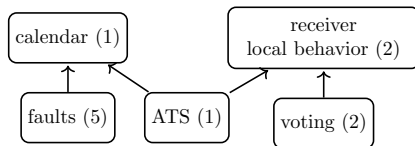
1. Calendar automata
 - ▶ modeling message passing, including details like real time, message delay, buffering, . . .
2. Symbolic fault injection
 - ▶ injecting faults into the model in a way that is decoupled from local node behavior
3. Abstract transition systems
 - ▶ strengthening invariants needed to verify properties of the system without having to abstract actual system behavior

Symbolic Fault Injection

- ▶ typical approach to modeling faults: add a new state variable to for each node, indicating fault state. The node's logic then acts on this value.
- ▶ fault injection:
- ▶ divorce fault behavior from node behavior by injecting faults at the channel level
- ▶ faulty values are represented as uninterpreted constants
- ▶ faulty values are constrained depending on the fault type
- ▶ separation of concerns allows us to reason about the fault model in isolation from local node behavior

Verification: Lemma Diagram

Our verification of the case study proceeds by k -induction, using SAL [3]. We strengthen the inductive hypothesis using hand-crafted lemmas which are organized as follows:



Modularity: Variant Models

To assess how modular our modeling framework is, we experimented on our case study, adding or changing features of the model and measuring how much effort was required to make the changes and verify the result.

1. Omissive asymmetric faults
2. Time triggered message passing
3. Mid-value selection voting

Omissive Assymmetric Faults

Consider changing the fault model:

- ▶ Removing all faults from the model is easy: we simply set the maximum weighted sum in the maximum fault assumption to zero.
- ▶ Adding a new fault mode is more work, but is still modular:
- ▶ we add a new uninterpreted function definition for omissive asymmetric faults
- ▶ we modify the fault type definition in SAL
- ▶ finally, we define the effect of this new fault type on the calendar

	Definitions (58 total)	Invariants (11 total)	Invariant classes (5 total)
Omissive Asymmetric Faults	1 new, 2 modified	2 modified	faults

References

- [1] P. Thambidurai and Y.K. Park. *Interactive consistency with multiple failure modes*. Symposium on Reliable Distributed Systems, 1988.
- [2] P. Lincoln and J. Rushby. *A formally verified algorithm for interactive consistency under a hybrid fault model*. In Fault Tolerant Computing Symposium 23, 1993.
- [3] SAL. Computer Science Lab, SRI International.
<http://sal.csl.sri.com>.