Diagnosing a Failed Proof in Fault-Tolerance: A Disproving Challenge Problem

Lee Pike[1] (presenting), Galois Connections
leepike@galois.com

Paul Miner and Wilfredo Torres-Pomales
NASA Langley Research Center
{p.s.miner, w.torres-pomales}@larc.nasa.gov

August 16, 2006

[1]Most of this work was performed while this author was employed at the NASA Langley Research Center Formal Methods Group.

Goals and Non-Goals

Goals:

1. Provide a medium-sized real-world case-study/challenge problem for disproving.
2. Describe the domain and tell a bit of the story about how this bug arose.
3. Describe (what the authors take to be) the best current approach.

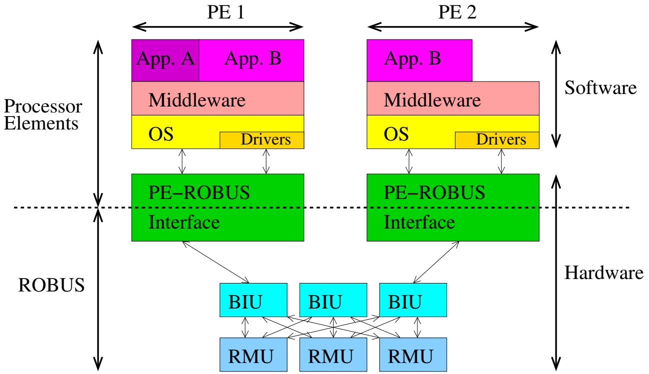
Non-Goals:

1. Present the protocol in detail (paper and formal (dis-)proof artifacts are available).
2. Present new results in disproving research.

Fault-Tolerant Bus Architectures

- Buses for fly- and drive-by-wire applications.
- Failure rates must be approx. 10^{-9}/hour of operation for fly-by-wire.
- Examples:
    - Time-Triggered Architecture (TTTech)
    - FlexRay (auto consortium)
    - SafeBus (Honeywell)
    - SPIDER (NASA Langley)

The SPIDER Architecture



SPIDER

(Scalable Processor-Independent Design for Extended Reliability)

ROBUS

(Reliable Optical bus)

ROBUS Protocols

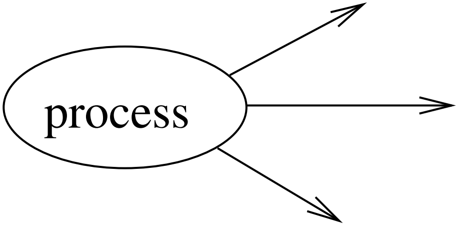
- Interactive Consistency Protocol (IC Protocol)
- Clock Synchronization Protocol
- Distributed Diagnosis Protocol
- Startup/Restart Protocol
- Reintegration Protocol

These are distributed, fault-tolerant, real-time protocols with complex interdependencies.

The Hybrid Fault Model[2]

Let V be the uncorrupted message to be sent.

- *Good* processes send all messages correctly.
- *Benign* processes send only benign messages.
- *Symmetric* processes send the same arbitrary message.
- *Asymmetric* processes send arbitrary messages.

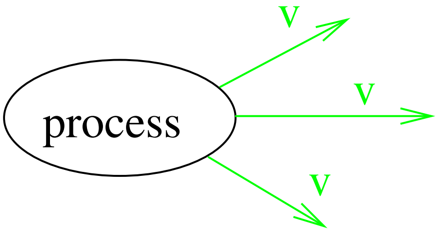


[2]Thambidurai and Park. Interactive consensus with multiple failure modes. *7th Reliable Distributed Systems Symposium*, 1988.

The Hybrid Fault Model[2]

Let V be the uncorrupted message to be sent.

- *Good* processes send all messages correctly.
- *Benign* processes send only benign messages.
- *Symmetric* processes send the same arbitrary message.
- *Asymmetric* processes send arbitrary messages.

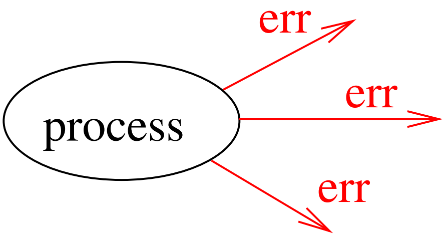


[2] Thambidurai and Park. Interactive consensus with multiple failure modes. *7th Reliable Distributed Systems Symposium*, 1988.

The Hybrid Fault Model[2]

Let V be the uncorrupted message to be sent.

- *Good* processes send all messages correctly.
- *Benign* processes send only benign messages.
- *Symmetric* processes send the same arbitrary message.
- *Asymmetric* processes send arbitrary messages.

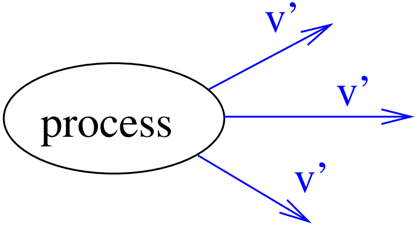


[2]Thambidurai and Park. Interactive consensus with multiple failure modes. *7th Reliable Distributed Systems Symposium*, 1988.

The Hybrid Fault Model[2]

Let V be the uncorrupted message to be sent.

- *Good* processes send all messages correctly.
- *Benign* processes send only benign messages.
- *Symmetric* processes send the same arbitrary message.
- *Asymmetric* processes send arbitrary messages.

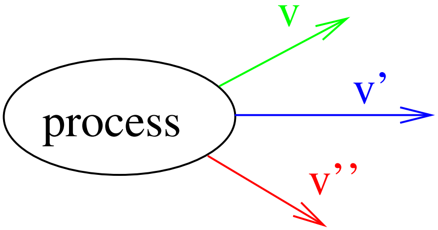


[2]Thambidurai and Park. Interactive consensus with multiple failure modes. *7th Reliable Distributed Systems Symposium*, 1988.

The Hybrid Fault Model[2]

Let V be the uncorrupted message to be sent.

- *Good* processes send all messages correctly.
- *Benign* processes send only benign messages.
- *Symmetric* processes send the same arbitrary message.
- *Asymmetric* processes send arbitrary messages.



[2]Thambidurai and Park. Interactive consensus with multiple failure modes. *7th Reliable Distributed Systems Symposium*, 1988.

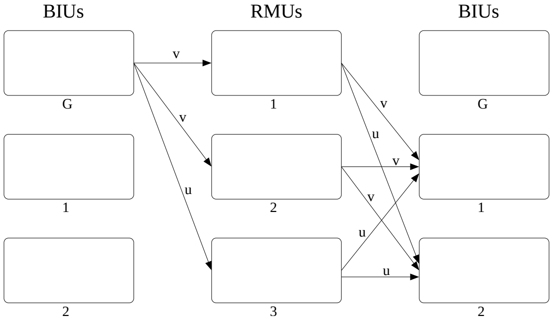
Dynamic Maximum Fault Assumption for the IC Protocol

The kinds and number of faults under which the system is hypothesized to behave correctly.

Informally,

1. Good BIUs trust strictly more good RMUs than symmetrically or asymmetrically-faulty RMUS, and
2. Either no good RMU trusts an asymmetrically-faulty General, or no good BIU trusts an asymmetrically-faulty RMU.

The SPIDER IC Protocol



IC Protocol Correctness

The protocol is designed to reliably passes data from a designated BIU (the "General") to the other BIUs.

- **Agreement**: All good BIUs compute the same value.
- **Validity**: If the General is good and broadcasts message v, then the value computed by a good BIU is v.

What must be specified to prove correctness?

- System assumptions (i.e., guarantees of the other protocols).
- A model of execution (to model synchronous message-passing).

Goal: Prove

$$\text{Let E} = \text{exec(protocol model) in}$$
$$\text{MFA \& system assumptions} \Longrightarrow \text{(Agreement(E) \& Validity(E))}$$

Bug Origins

Incomplete/changing/ambiguous system requirements!

- Two coordinating domain experts (Paul Miner and Wilfredo Torres-Pomales).
- Ambiguity of the dynamic fault model and granularity of timing.
- Changing fault assumptions and new reintegration requirements.

A Dastardly Bug

Why is this a "significant" bug?

- The bug requires two simultaneous Byzantine faults to occur (permitted by the MFA).
- Thus, it would assuredly have been overlooked in fault-injection testing.

Wilfredo discovered the bug by inspection. But...

- Wilfredo is a one the world's handful of experts in the domain.
- The ROBUS is *just* small enough that a single engineer can understand the whole design.

Our (Conventional) Approach

- There was an on-going effort to verify the ROBUS protocols via theorem-proving (PVS).
- Thus, we wanted to see if we could “recreate” the bug formally by discharging all branches of its proof except for the buggy case.
- We could! But, it left us with a bit of a mess. . .

The Mess (in PVS)

(Irrelevant formulas hidden)

```
[-1]  good?(r_status!1(r!1))
[-2]  asymmetric?(b_status!1(G!1))
[-3]  IC_DMFA(b_status!1, r_status!1, F!1)
[-4]  all_correct_accs?(b_status!1, r_status!1, F!1)
  |-------
[1]   trusted?(F!1`BR(r!1)(G!1))
[2]   declared?(F!1`BB(b2!1)(G!1))
{3}   (FORALL (p_1: below(R)):
         (trusted?(F!1`RB(b1!1)(p_1)) =>
            NOT asymmetric?(r_status!1(p_1))))
       &
       (FORALL (p_1: below(R)):
          (trusted?(F!1`RB(b2!1)(p_1)) =>
             NOT asymmetric?(r_status!1(p_1))))
[4]   declared?(F!1`BB(b1!1)(G!1))
[5]   robus_ic(b_status!1, r_status!1,
               F!1`BB(b1!1)(G!1), F!1`RB(b1!1))
              (G!1, msg!1, b1!1)
       =
       robus_ic(b_status!1, r_status!1,
                F!1`BB(b2!1)(G!1), F!1`RB(b2!1))
               (G!1, msg!1, b2!1)
```

Getting a Counterexample

The first author of this paper was competent at PVS but not a domain expert at the time of the proof. Left to his own devices, he would not have been able to tell if the undischarged subgoal was the result of

- going down a blind alley in the proof,
- an invariant that was too weak,
- a problem with the formal model,
- a bug in the protocol.

But a model-checker (SAL) will give a counterexample...

The Undischarged Sequent as a Safety Property

```
counterex: THEOREM SYSTEM |-
  G(  (pc = 4 AND
       r_status[1] = good AND
       G_status = asymmetric AND
       IC_DMFA(r_status, F_RB, F_BR, G_status) AND
       all_correct_accs(r_status, F_RB,
                        G_status, F_BR, F_BB))
    =>
      (F_BR[1] = trusted OR
       F_BB[2] = declared OR
       (FORALL (r: RMUs): F_RB[1][r] = trusted =>
          r_status[r] /= asymmetric AND
        FORALL (r: RMUs): F_RB[2][r] = trusted =>
          r_status[r] /= asymmetric) OR
       F_BB[1] = declared OR
       robus_ic[1] = robus_ic[2]));
```

Shortcomings of Our Approach

- Two models of the protocol, assumptions, and requirements built, one in PVS and one in SAL (no automated translation).
- Manual instantiation of parameters.
- Manual translation of correctness conditions (HOL $\rightarrow$ LTL).

The Challenge

From a parameterized specification of the protocol (from which a general proof can be obtained), provide a concrete instance of the bug in a way that requires as little effort from the user as possible.

Success Criteria

An approach that is simpler and more efficient than ours:
The *upper bound* on effort is the time required for a moderately-skilled theorem-prover with some domain expertise to uncover the error by inspecting the failed proof in a mechanical theorem-prover.

Possible Approaches(?)

Warning: Half-baked speculation.

- Quickcheck or a FOL automated prover (for counterexample generation).
- Automated translator from a theorem-prover to a model-checker.
- Automated parameter interpretation.

Bonus Challenge

But what we'd *really* like is...

To prove the correctness of the parameterized protocol in the first place in a more automated way.

What makes this problem hard?

- Parameterized design.
- Nontrivial mathematical reasoning.
- Nondeterminism introduced by modeling faults (both the kind of fault and the when they occur).

But don't take my word about their difficulty. . .

- Pat Lincoln and John Rushby describe a flawed Oral Messages algorithm (uncovered via theorem proving) and a verified fixed algorithm (CAV, '93).
- William Young compares Interactive Consistency (IC) in PVS vs. ACL2 – IC had been proposed as a benchmark for *interactive* theorem-proving (Conference on Computer Assurance, '96).
- John Rushby, Shmuel Katz, and Pat Lincoln themselves *incorrectly* specified a Group Membership algorithm. The error was spotted and revised, but a formal proof of the revised algorithm was not discovered by Katz, Lincoln, and Rushby for a year (CAV, 2000).

The Future

A (mostly) automated proof of this protocol would be a boon to fault-tolerant system designers and demonstrate that what is still considered a difficult interactive proving challenge can be completed much more easily.

What will the future look like?

- Satisfiability modulo theories (SMT) provers?
- Specialized tactics/proof strategies?
- Specialized provers for fault-tolerant protocols?
- Parameterized model-checking?
- Some combination thereof?

Additional Information

Specs & (Dis-)Proofs in PVS and SAL

`http://www.cs.indiana.edu/~lepike/pub_pages/disprove.html`

Google: Pike disproving

SPIDER Website

`http://shemesh.larc.nasa.gov/fm/spider/`

Google: fm program spider

Appendix.

IC Protocol Description

1. The General, G, broadcasts its message, v, to all RMUs.
2. For each RMU, if it receives a benign message from G, then it broadcasts the special message *source error* to all BIUs. Otherwise it relays the message it received.
3. For each BIU b, if b has declared G, then b outputs the special message *source error*. Otherwise, if b received a benign message from an RMU, then that RMU is accused. b performs a majority vote over the values received from those RMUs it trusts. If no majority exists, *source error* is the result; otherwise, the majority value is the result.

IC Maximum Fault Assumption

1. $|GR \cap \mathtt{T}_b| > |SR \cap \mathtt{T}_b| + |AR \cap \mathtt{T}_b|$;
2. $G \in AB \cap \mathtt{T}_r$ implies $|AR \cap \mathtt{T}_b| = 0$.