SPIDER

A Layered Assurance Perspective: Lessons from the Formal Analysis of Fault-Tolerant Systems

Paul S. Miner

p.s.miner@nasa.gov

5th Layered Assurance Workshop

5 December 2011

fault-tol·er·ant \'fȯlt-'täl(- ə)-rənt\
adj : able to function in the absence of a major component

Outline

- Background on Safety-Critical Systems
- Overview of SPIDER
- Generalizations for reuse
- Lessons:
  - Mechanism vs. policy
  - Combining architectures
  - Assurance layers vs. architecture layers

Safety-Critical Systems Focus

Emphasis on

Integrity

- In many cases, (known) loss of function is preferable to malfunction

Availability

- Need to preserve capability to provide critical functions

Generally not concerned with **Confidentiality**

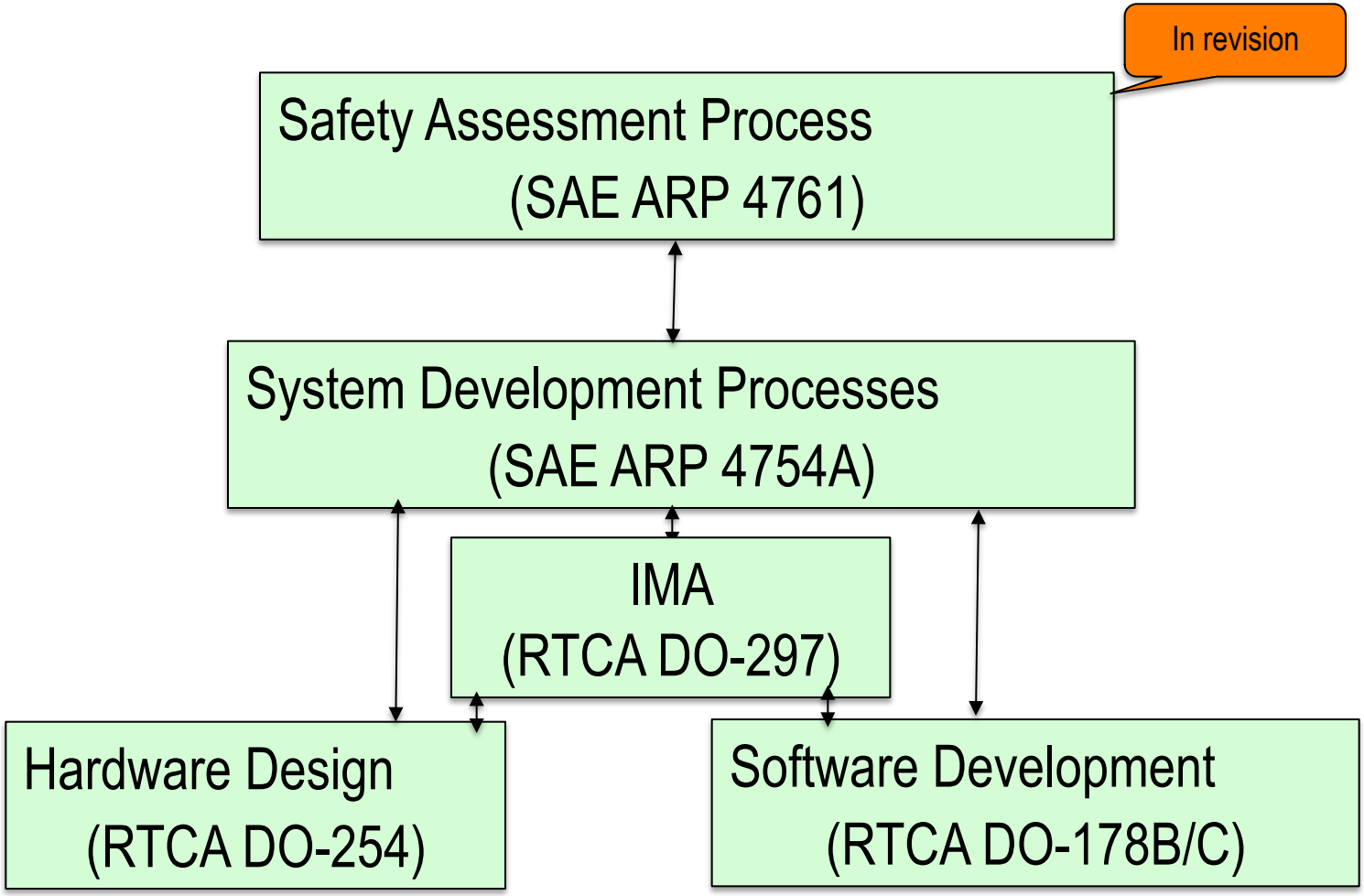
What is a Safety-Critical System?

- The failure of a *safety-critical system* has the potential to cause the loss of life or serious injury
- Examples include:
  - Aircraft
  - Automobiles
  - Trains and Rail systems
  - Medical Devices

Quantifying Safety-Critical

- For systems onboard transport aircraft, the FAA requires that the catastrophic failure rate be no higher than 10^{-9} per average flight hour
- How low is that?
  - 10^{9} hours is 114,156 years
  - Odds of being hit by lightning in a year: 1 in 240,000
  - Odds of dying in a car wreck per trip: 1 in 4 million

Relevant Civil Aviation Standards



Integrated Modular Avionics (IMA)

- Integrated architectures provide (computational) resources for several distinct aircraft functions
  - Aircraft functions have different levels of criticality determined by the potential severity of failure effects
  - If functions of different criticality share (computational) resources, then architecture must manage resources
    - Otherwise, failures of non-critical functions can prevent critical functions from accessing necessary resources
- Correct operation of IMA platform resource management is at least as critical as the most critical hosted application

ARINC 653

- Real-Time Operating System (RTOS) standard for safety-critical systems
- Provides for robust partitioning of computational resources on a single processor
  - Enforces guaranteed (scheduled) processor time allocation
  - Preserves integrity and availability of allocated memory
- Several commercially available options
  - Green Hills, Wind River, LynuxWorks, …
- Provides protection against some software faults in other applications
  - No protection against erroneous results propagating through defined interfaces
  - No protection against hardware or network failures

Safety-Critical Network Partitioning

- For safety-critical systems, we also need guaranteed communication channels between redundant processing sites
- Protection against random hardware faults
- Nodes on the network can fail in arbitrary ways
- Communication mechanism must provide guarantees of communication integrity, throughput, and latency, even if some attached devices are actively misbehaving

SPIDER Case Study

- Design part of a Fault-Tolerant Integrated Modular Avionics (FT IMA) architecture concept
  - Fault-tolerance is inherently complex
  - System description is compact
- Case study applied to the Reliable Optical Bus (ROBUS) of the Scalable Processor-Independent Design for Electromagnetic Resilience (SPIDER)
- ROBUS operates as a fault tolerant time division multiple access (TDMA) broadcast bus

SPIDER Contributors

Architecture and protocol development - Paul Miner, Wilfredo Torres, Mahyar Malekpour

Hardware design and implementation - Wilfredo Torres, Mahyar Malekpour

Formal verification – Paul Miner, Jeff Maddalon, Alfons Geser (was NIA, now HTWK Leipzig), Lee Pike (now with Galois)

Design Objective

- FT-IMA Architecture proven to survive a *bounded* number of ***physical*** faults
  - Both permanent and transient
  - Must survive ***Byzantine*** faults
- Capability to survive or quickly recover from massive correlated transient failure (e.g., in response to HIRF, solar flare, etc.)

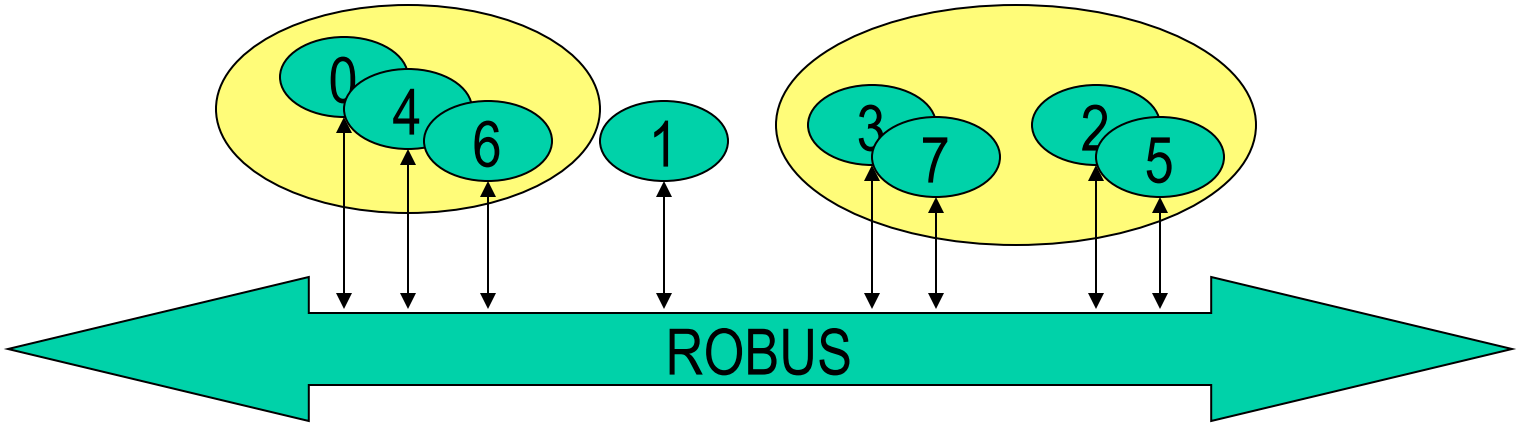
ROBUS Characteristics

- All good nodes agree on communication schedule
  - Originally bus access schedule statically determined
    - Similar to SAFEbus (ARINC 659), TTP/C
  - Architecture now supports on-the-fly schedule updates
    - Similar to FTPP
    - Implemented in ROBUS-2
- Some fault-tolerance functions provided by Processing Elements (PE)
  - Similar to FT-Layer in Time-Triggered Architecture
  - Can either be in hardware or middleware layer
- Processing elements need not be uniform
  - Support for some dissimilar redundancy

SPIDER

Logical view of SPIDER

(Sample Configuration)



ROBUS Requirements

- All fault-free PEs observe identical message sequences
  - If the source is also fault-free, they receive the message sent
- ROBUS provides periodic synchronization messages
  - The PEs are synchronized relative to this
- ROBUS provides correct and consistent ROBUS diagnostic information to all fault-free PEs

All protocols analyzed with respect to the same maximum fault assumption

Other Requirements?

- Primary focus is on fault-tolerance requirements
  - Other requirements deliberately unspecified
    - Message format/encoding
    - Performance
  - These are implementation dependent
- Product Family
  - Capable of a range of performance
  - Trade-off performance and reliability
  - Formal analysis valid for any instance

Redundancy Management (RM)

- Redundancy management techniques are tailored to specific sources of faults
  - Examples include:
    - Algebraic encoding for single bit/byte/message errors
      - Parity, CRC, SEC/DEC encoding, etc.
      - Used to detect and/or recover from naturally occurring transient disruptions
    - State-machine replication
      - Self-checking pair, TMR with majority vote, etc
      - Used to survive faults that result in an incorrectly computed result
      - Requires that redundant channels *agree* on data prior to computation
- *There is no Universal fault-tolerance solution*

Factors influencing RM Strategy

- Cost of system failure
  - Loss of life/mission/payload
  - Economic impact (e.g., power grid, telecom, financial markets, etc.)
- Fail-Operational vs. Fail-Stop/Fail-Safe
  - If fail-op, is degradation of performance okay?
  - Do we need fault diagnosis? Or is it okay to mask effects?
- Exposure to risk (e.g., duration of mission/critical stages)
- Likelihood of various classes of faults
  - *Do you allow a potential single point of failure if its probability of occurrence is sufficiently low?*

Allocate redundancy carefully

http://xkcd.com/970/

Failures contained by ROBUS

ROBUS must tolerate

- A bounded number of internal hardware failures (including Byzantine)
- Arbitrary failure in any attached PE
  - Hardware or software
- Cannot tolerate design error within ROBUS
  - Design error in ROBUS could result in total loss of function

How to achieve?

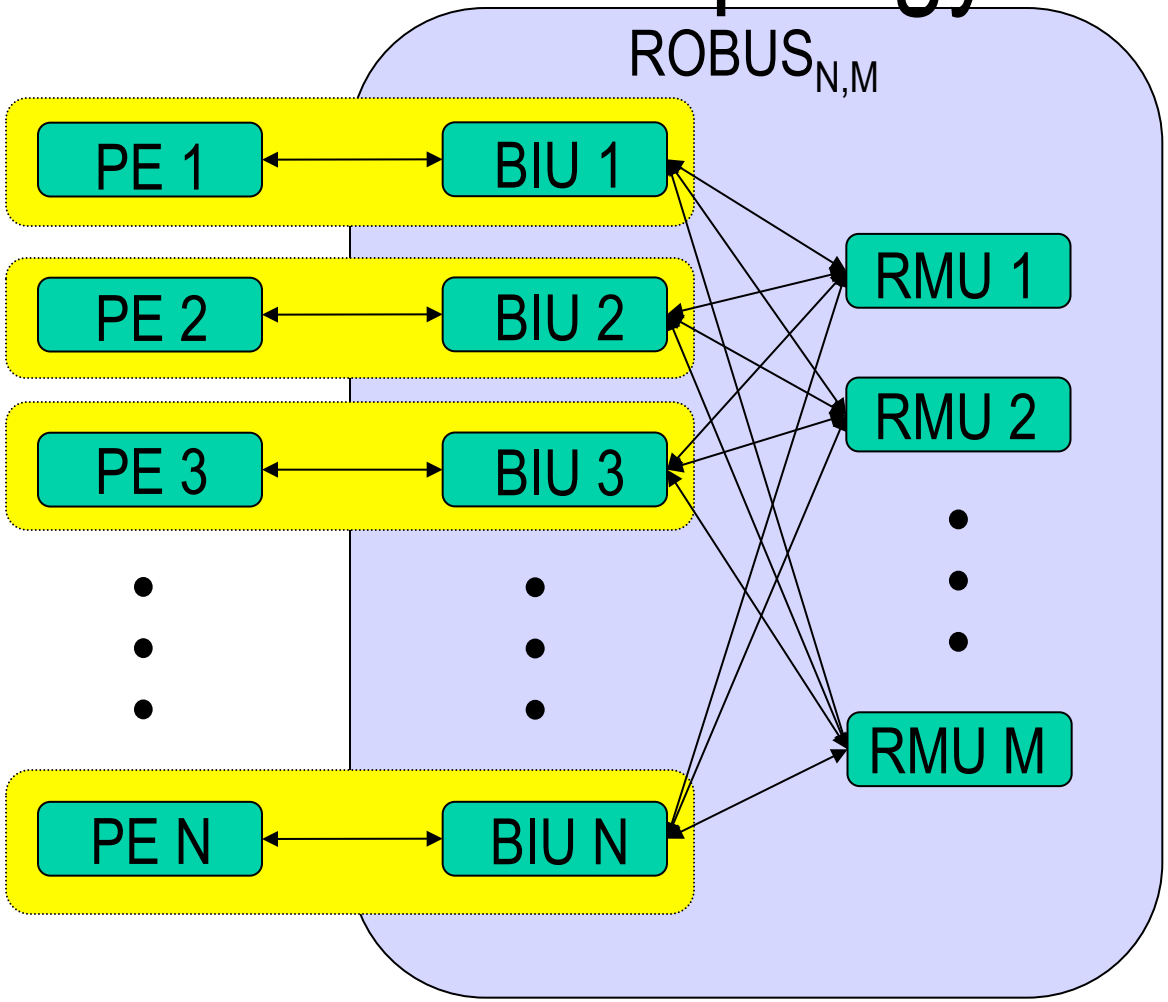
- System Architecture
- Markov analysis calculates Pr(enough good hardware)
- Design assurance methods provide (proven in PVS):

  |- enough good hardware => correct operation

ROBUS Protocols

- Interactive Consistency (Byzantine Agreement)
  - loop unrolling of classic Oral Messages algorithm
  - Inspired by Draper FTP
- Clock Synchronization
  - Adaptation of Srikanth & Toueg protocol to SPIDER topology
  - Corresponds to Davies & Wakerly approach
- Distributed Diagnosis (Group Membership)
  - Initially adapted MAFT algorithm to SPIDER topology
    - Depends on Interactive Consistency protocol
  - Verification process suggested more efficient protocol
    - Improved protocol due to Alfons Geser
    - Suggested further generalizations

ROBUS Topology



ROBUS Design Evolution

- ROBUS 1
  - Static communication schedule
  - Sequential communication
    - Throughput: 1 message in the system at a time
  - Full-configuration startup only
  - Reconfiguration only through degradation
- ROBUS 2
  - Dynamic schedule update
  - Pipelined communication
    - Maximum throughput: 1 message per tick
  - Enhanced error detection
  - Variable-configuration startup
  - Reconfiguration through degradation and reintegration
  - Automatic restart

ROBUS-2 Prototype

- Laboratory implementation of ROBUS-2
- VHDL Design of ROBUS-2 released under NASA Open Source Agreement (NOSA)

http://ti.arc.nasa.gov/opensource/projects/robus-2/

Byzantine Faults

- Characterized by asymmetric error manifestations
  - Different manifestations to different fault-free observers
  - Including dissimilar values
- Can cause redundant computations to diverge
- Can cause distributed decisions to conflict
- If not properly handled, a single Byzantine fault can defeat several layers of redundancy
- Many architectures neglect this class of fault
  - Assumed to be rare or impossible

Byzantine faults are real

- Several examples cited in *Byzantine Faults: From Theory to Reality,* Driscoll, et al. (SAFECOMP 2003)
  - Byzantine failures nearly grounded a fleet of large aircraft
  - Quad-redundant system failed in response to a single fault
  - Typical causes are faulty transmitters (resulting in indeterminate voltage levels at receivers) or faults that cause timing violations (so that multiple observers perceive the same event differently)
- Heavy Ion fault-injection results for TTP/C (Sivencrona, et al.)
  - More than 1 in 1000 of observed errors had asymmetric (Byzantine) manifestations
- STS-124 pre-flight (May 2008)
  - http://www.nasaspaceflight.com/2008/05/sts-124-frr-debate-outstanding-issues-faulty-mdm-removed/

First Picture of a Byzantine Fault?

At 12:12 GMT 13 May 2008, a NASA Space Shuttle was loading hypergolic fuel for mission STS-124 when a 3-1 split of its four control computers occurred. Three seconds later, the split became 2-1-1. During troubleshooting, the remaining two computers disagreed (1-1-1-1 split). ***Complete system disagreement.*** But, none of the computers or their intercommunications were faulty! The ***single fault*** was in a box (MDM FA2) that sends messages to the 4 computers via a multi-drop data bus that is similar to the MIL STD 1553 data bus. This fault was a simple crack (fissure) through a diode in the data link interface.

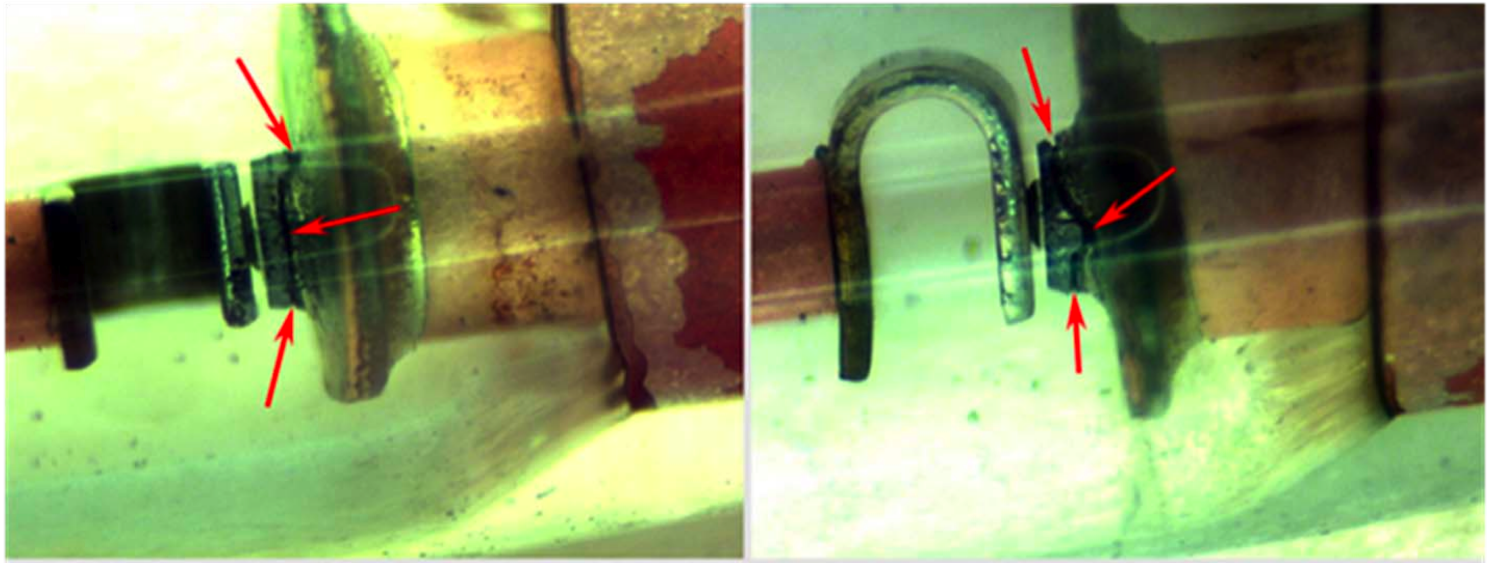


Figure 1. Two views (90 degrees apart) of a fissure that appears to go through the silicon dice - Red arrows.

From Kevin Driscoll's Keynote Presentation at SAFECOMP 2010 (with permission)

Hybrid Fault Assumption

- Architectures designed assuming only Byzantine faults can be brittle
  - David Powell, *Failure Mode Assumptions and Assumption Coverage*, FTCS-22, 1992
- We must handle Byzantine faults
  - But, other failure modes *are* more common
- Systems designed using hybrid fault models gracefully accommodate either a few Byzantine faults or combinations of several less severe faults
  - Avoids assumption coverage difficulties

Hybrid Fault Hypothesis

- Byzantine faults are real, protocols must be designed against this worst case fault manifestation
- To tolerate *f* Byzantine faults requires
  - 3*f* + 1 independent fault containment regions (FCR),
  - 2*f* + 1disjoint communication paths between every pair of FCRs, and
  - *f* + 1 stages of communication
- Easier to handle faults occur more frequently
  - Hybrid fault models established for various manifestations of misbehavior
    - Transmission (incorrect value) vs. omission (fail-silent, fail-stop),
    - Symmetric (same to all receivers) vs. asymmetric
    - Multiple distinct value asymmetry vs. at most one value
  - Omission faults can be ignored, provided there is still a good source
  - Asymmetric/transmission combination includes Byzantine manifestations
- *Classification is a function of state of all receivers!*

Unified Consensus Protocol

- All SPIDER/ROBUS fault-tolerance protocols may be realized using different instances of a single abstract protocol
  - Generalization of Davies & Wakerly [IEEE ToC 1978]
  - Abstracted protocol consists of a cascade of data exchanges with a middle value selection vote at each stage
- Formal analysis using PVS presented at FORMATS-FTRTFT September 2004
  - Using Thambidurai & Park hybrid fault assumptions
    - Good, Benign, Symmetric, Asymmetric
  - Paper and PVS models available from SPIDER web site
    - http://shemesh.larc.nasa.gov/fm/spider/
- Subsequently generalized to encompass weaker fault assumptions and more flexible voting strategies
  - Added convergent voting for approximate agreement

Generalization Goal

- Reusable collection of formal models for analysis of distributed consensus protocols
  - That provide correct service in the presence of malfunction/ misbehavior of a subset of participants
    - Considering various severities of misbehavior
  - This class of protocols spans Aviation Safety, Airspace, and Exploration (as well as several non-NASA challenges)
- Current Status
  - PVS Fault-Tolerance Library covering synchronous consensus protocols under a hybrid fault model
    - http://shemesh.larc.nasa.gov/fm/ftp/larc/PVS-library/pvslib.html

Some Distributed Consensus Protocols

- Clock Synchronization
- Group Membership (Distributed Diagnosis)
  - Need to disambiguate faults in presence of imprecise information
- Interactive Consistency/Source Congruency
- Reintegration/transient fault recovery
- Start-up/Re-start

Want all of these in presence of specified number of faults

Distributed Consensus Properties

With respect to some critical {event/data/state/decision} *x*

Validity

All correctly behaving participants perception of *x* is consistent with the correct value of *x* (within some error tolerance)

Agreement

All correctly behaving participants have a consistent perception of *x* (within some error tolerance)

Distributed consensus properties provide a foundation for compositional verification

Communication Fault Model

- A communication stage consists of a set of sources transmitting local values to a set of receivers
- At any stage of communication a receiver may *ignore* a message from a given source if the message fails in-line checks (including absence of message) or if prior diagnosis indicates the source to be untrustworthy
- The receivers make a fault-tolerant choice from among the set of messages received (ignored messages are not considered)
- Classification of fault manifestations determined by global view of received messages (for each communication stage)
- The fault classification is of the source
- Used Azadmanesh & Kieckhafer's (IEEE ToC 2000) extensions to the Thambidurai & Park (1988 Reliable Distributed Systems) hybrid fault model
  - As extended by Weber (2006)

Informal Hybrid Fault Classification

Good: All receivers receive *correct* value

Benign (symmetric omission): Either all receivers receive *correct* value or all receivers *ignore* message

Asymmetric omission: Some receivers receive *correct* value while others may *ignore* message

Symmetric: All receivers receive same (possibly incorrect) value

Single value: All receivers that receive a message receive the same (possibly incorrect) message. Some receivers may *ignore* message.

Asymmetric transmission (Byzantine): At least two different receivers receive different messages

Single Stage Properties

Validity: If there are *enough correct* values received, then all *good* receivers select a value in the range of the *correct* values

Agreement Propagation: If there are *enough correct* sources and all *correct* sources agree, then all *good* receivers will agree

- Corollary of Validity

Agreement Generation: If there are no *asymmetric* sources, then all *good* receivers will agree

Convergence: If there are *enough correct* sources and *correct* sources agree within δ and *enough* common values averaged by any pair of *good* receivers, then the value selected by all *good* receivers will agree within some $f(\delta, \varepsilon) < \delta$

Multistage Properties

Validity: If there are *enough good* sources at every stage, then all *good* receivers select a value in the range of the *good* sources from the first stage

- Follows from induction on stages and single-stage validity result

Agreement Propagation: If there are *enough good* sources at every stage, and all *good* sources for the first stage agree, then all *good* receivers will agree

- Follows from induction on stages and single-stage agreement propagation
- Also a corollary of multi-stage validity

Agreement Generation: If there exists a stage with no *asymmetric* sources and there are *enough good* sources at every subsequent stage, then all *good* receivers will agree

- Follows from single-stage agreement generation and multi-stage agreement propagation

Convergence: If there are *enough good* sources at every stage and there exists at least one converging stage, then the multi-stage exchange converges

- Errors introduced in communication (unavoidable for synchronization) and asymmetric faults may limit degree of convergence

ROBUS-2 Consensus Protocols

- Two-stage agreement generation (at least one stage without asymmetric fault):
  - Clock synchronization
  - Group membership (distributed diagnosis)
  - Schedule update
  - Interactive consistency / source congruency

  All assuming synchrony, and sufficient agreement on schedule and diagnostic state
- One-stage agreement propagation
  - Reintegration / transient fault recovery
  - Communication schedule must ensure that all critical state information is periodically distributed in an agreement propagation stage
- Start-up / Re-start
  - This is a special two-stage agreement generation exchange for both synchronization and diagnostic state
  - Assumes (very) coarse initial synchrony

But, …

- This collection of models not universal
- Different paradigms do not integrate seamlessly
- Layered assurance is not the same as layers of protection

Engineering of Complex Systems

"As Eads, Flad, and Pfeifer knew, the essence of sound engineering lay in clearly stating the assumptions upon which calculations are based so that they may be checked at all times for lapses in logic and other errors. It is thus imperative that engineering premises be set down clearly, and that the calculations that follow be systematically and unambiguously presented, so that they may be checked by another engineer with perhaps a different perspective on the problem."

Henry Petroski, "Engineers of Dreams", p. 44, Alfred A. Knopf, New York, 1995.

SPIDER

An assumption will remain valid only until you come to depend on it*.

* http://www.ece.mtu.edu/faculty/rmkieckh/Kieckhafer-top-ten.htm (version 9.1; law 4.2)

Independence Assumption

- All redundancy mechanisms assume faults affect redundant pieces (e.g. bits, messages, channels) independently
- System must be engineered to satisfy this assumption (with respect to the covered class of faults)
- For systems based on state-machine replication, there is the notion of a Fault-Containment Region (FCR)
  - A fault in one FCR must not trigger faulty behavior in another
  - Conservative realizations of FCRs provide for both spatial and electrical isolation
  - Less conservative realizations have exhibited violations of independence

As far as the laws of mathematics refer to reality, they are not certain; and as far as they are certain, they do not refer to reality.

– Albert Einstein

Caveat: No such thing as perfect fault containment in physical domain

Further Generalizations?

- Asynchronous architecture
- Different architectural decompositions

Fault-Tolerant Avionics SOA:

Competing Philosophies

Asynchronous

- Common in primary flight control systems
- Unsynchronized, but common frame rate
  - Coarse synchronization needed for mode change (i.e., exact agreement)
- Imprecision of sensor data
- Approximate Agreement w/ Threshold voting
  - Incorrect threshold is a common source of failure (diagnostic ambiguity)

Synchronous

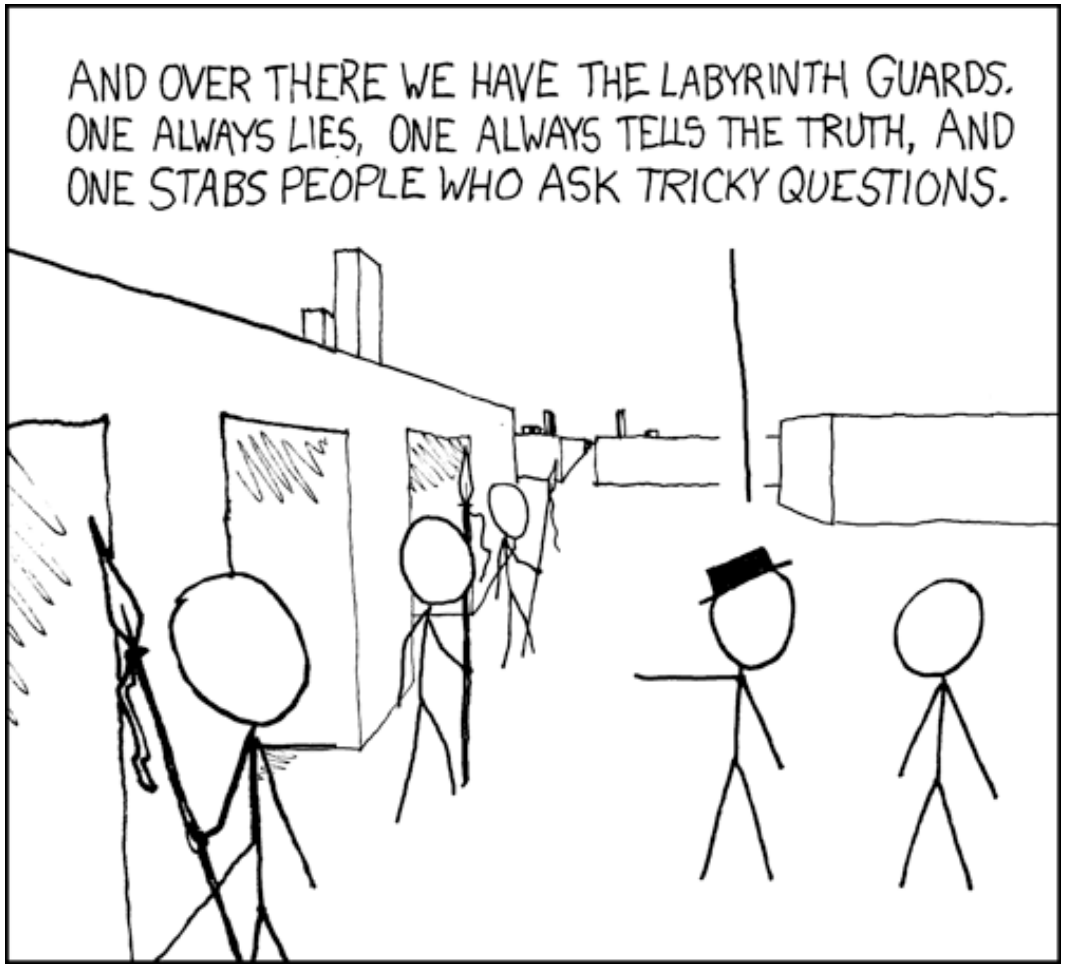
- Common in flight management systems
- Precise Synchrony
- State replication
- Reliable Data Distribution
- Exact agreement / Majority voting
- Approximate agreement only for synchronization

Differences in Architecture

- ROBUS guarantees agreement at local fault-tolerant network layer
  - Validity of message content enforced at middleware layer or above
- Self-checking architectures are also common
  - Validity enforced at every stage of communication
  - Agreement enforced at middleware or above
- How do we compose architectures with different layering philosophies?

Closing Thoughts

- System assurance argument not readily apparent through multiple mechanisms which comprise layers of protection
  - Ex. Fault Containment vs. Error Containment (self-check vs. masking)
- Layers of assurance rely on interfaces to regulate propagation of errors between containment zones
- Assumptions/guarantees are critical in determining how layers interact (implicit/explicit interface constraints)
- Emergent behavior of policy implemented by various mechanisms is not necessarily apparent (non-compositional)
  - Caveat: No such thing as perfect containment in physical domain
- Physical topology /= logical component boundaries /= protection mechanisms



Questions?

Downloaded from http://xkcd.com/246/

Backup

All models are wrong but some are useful

George Box

System Protection Goals

Attackers look for the weakest link, so not just strong barriers, but …

strong barriers *and* redundancy

Mechanisms for Fault-Tolerance

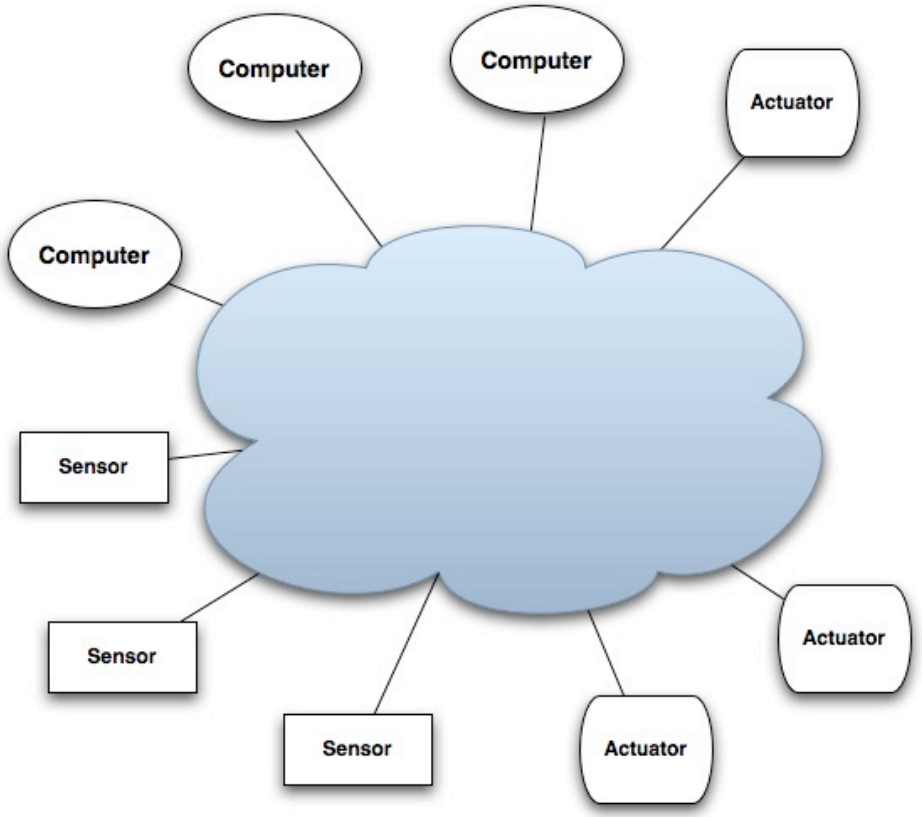
- Redundancy
  - Without some form of redundancy you cannot detect or recover from a fault
- Redundency
  - The only thing guaranteed by introducing redundancy is that the probability of a fault occurring increases -- Lala, et al.
- Radundancy
  - There is always an upper bound on the number of faults that a given redundancy mechanism can tolerate

Redundancy only works if the redundant elements fail *independently*

Gross Generalizations

- Aircraft safety depends on a networked collection of {sensors, computers, actuators, pilots, …}
- Airspace safety depends on a networked collection of {sensors, computers, actuators, pilots, controllers, …}
- The advent of digital networks and computers has fundamentally altered the landscape for the safety assessment of these systems
  - Roles and responsibilities are migrating across traditional system decomposition boundaries
    - Between humans and automation
    - Between aircraft and ground
    - Between systems on-board the aircraft
- One essential safety-enabling property for these systems is consensus amongst the various networked entities

Over-simplified {aviation} network



Pilots

Controllers

…