NITRD meeting on The New Clockwork for Time-Critical Systems, Hyatt Baltimore 25, 26 October 2012, based on Planning Meeting for the National Workshop on The New Clockwork for Time-Critical Systems 11, 12 June 2012 at Holiday Inn, Fairborn (Dayton), Ohio 45324

Timing Robustness and Fault-Tolerance

John Rushby

Computer Science Laboratory
SRI International
Menlo Park CA USA

My Interest: Critical Systems

- Require very low rates of failure
  - Hence, redundancy, which adds complexity
- And strong assurance
  - Hence, preference for simplicity
- Conflict often resolved through strong assumptions
  - e.g., independent failures, synchronous networks
  - A lot of engineering needed to justify those
- Roughly, the challenge of resilience is to deliver reliability and assurance under weaker assumptions

Past and Future Problems

- I'll assert that problems concerning timing within a single system (which may itself be distributed and fault tolerant) are largely solved
    - Even though the state-of-the-art has not penetrated all application sectors
- For safety-critical systems, we generally build on a synchronous substrate
    - i.e., guaranteed bound on nonfaulty message delivery
- With nodes that fail independently
- Can then provide provably fault-tolerant clock synchronization
- And can employ time-triggered techniques on top
    - Eases fault tolerance, design, debugging
- Costly to develop, but now COTS (e.g., TTE)

Past and Future Problems

- I think the opportunities and challenges for the future arise when we relax former assumptions and expectations
- Underlying substrate is not synchronous, but is synchronized
  - e.g., with GPS, 1588
  - Can we still achieve 10^{-9}?
  - Can we do new things?
- Instead of a single system, we have a system of systems
- I'll think of a system as something that interacts with an environment and performs an independently useful function

Systems of Systems

- We put systems together (i.e., compose them), so that each becomes (part of) the environment of the other, because. . .
    - We actually want the combination of their capabilities (symmetrical use case)
    - Or one system needs some capabilities and it is simpler or cheaper to use another system to provide them, rather than develop a bespoke component (asymmetrical use case)
    - Or we didn't realize the consequences of our (often incremental) actions (accidental use case)
- And along with the benefits of composition, we sometimes get the flaws
    - E.g., car CD player has entire Linux inside;
      enables penetration of system and remote control of throttle/brakes (CarShark)

Emergent Misbehavior

- Complex systems can have failures not readily predicted from their components, interactions, or design
- Call this emergent misbehavior
- I'll save for another day the discussion whether these misbehaviors are merely unexpected or truly emergent
    - e.g., maybe some are due to downward causation
- But I think it can be useful to consider these failures as different in kind than the usual ones
- Examples
    - Feature interaction in telephone systems
    - West/East coast phone and power blackouts
    - 1993 shootdown of US helicopters by US planes in Iraq
    - Überlingen mid-air collision

Causes of Emergent Misbehavior

- I think they all come down to epistemic uncertainty
    - i.e., ignorance
- There is no complete and accurate description of the system simpler than the system itself
- But all our analysis and verification are with respect to abstractions and models, hence we are ignorant about the full set of system qualities
- More particularly, we may be ignorant about
    - The complete set of requirements we will care about in the composed system
    - The complete set of behaviors of each component
    - The complete set of interactions among the components

How to Eliminate or Control Emergent Misbehavior

- Identify and reduce ignorance, or equivalently improve the quality of our models
    - Is there a measure for doubt, for ignorance?
    - Economists tell me it would look like entropy

- Eliminate or control unanticipated behaviors and interactions
    - i.e., deal with the manifestations of ignorance

- Engineer resilience
    - i.e., adapt to the consequences of ignorance

- Let’s focus on the latter two, wrt. timing

Timing Robustness and Fault Tolerance In Systems of Systems

- Suppose we have large and variable delays in sensor and data exchange
- Potentially leading to instability and failures
- But we have system-wide synchronization (e.g., via GPS)
- i.e., system is synchronized but not synchronous
- Can dynamically create some of the attributes of time-triggered design
- e.g., using sparse time and π/Δ precedence
    - Events happen within π of each other, or at least Δ apart
    - Can then (but not otherwise) always sort out the temporal ordering of timestamped events
    - Parameters depend on synchronization, not delays

Challenges (1)

- Develop a comprehensive “theory” for this or other weakly synchronous approaches to this class of systems
  - i.e., my estimate of your state is accurate, and accurately timestamped, but (boundedly?) old
  - And sometimes messages are lost or arbitrarily delayed
- Or should we devolve to the asynchronous model?
  - With failure detectors
- Or to one of the partial synchrony models?
- These deal with various “degrees” of asynchrony, but do not contemplate that the system is synchronized
- Is there a decent programming model for any of these?
  - cf. Giotto for time-triggered

Challenges (2)

- Next, suppose we do not have a global source of synchronization (like GPS)
    - Or suppose that it is intermittent
- We want a method of fault-tolerant synchronization that
    - Does not assume a synchronous substrate (i.e., delays may be unbounded)
        - ⋆ Presumably need some additional assumptions
    - Is self-stabilizing (no special startup or reintegration)
    - Coexists and integrates with a global clock (i.e., GPS)
    - Tolerates a wide range and number of faults
    - Is high quality and degrades gracefully
- I know of no off-the-shelf algorithm with all these attributes

Challenges (3)

In systems with large and variable delays, should we. . .

- Try to develop control algorithms and fault-tolerance mechanisms that can cope with this?

- Or do synchronization and techniques like π/Δ precedence give us enough to use conventional algorithms?

A Thought Experiment

- Suppose that at some point in a system development I discern the need to make some part of it fault tolerant
- I could choose a strong (i.e., restrictive) fault model
- Then that might enable me to design a correspondingly simple algorithm to perform the fault tolerance
- Thus, I might have very few doubts about whether my algorithm is correct (wrt. its fault model)
- But I might have considerable doubts about whether the fault model will be valid in the real context of its deployment
- Alternatively, I could make few assumptions about the faults
- But then the mechanisms to tolerate those faults might take me into the world of complex adaptive systems
- Here I have fewer doubts about validity of the fault model, but more about correctness of my algorithm&implementation

Resilience

- There are just two sources of uncertainty (in the sense of doubt) in an assurance case
    - Epistemic: extent and accuracy of my knowledge about the system, its requirements, environment, etc.
    - Logic: validity of my reasoning about the correctness of the design wrt. requirements
    - cf. Validation and Verification (V&V)
- There is some opportunity to trade these (recall example)
- Traditionally, in critical systems, we have favored reducing logic doubt at the expense of epistemic doubt
    - e.g., no adaptive systems in flight control
- Resilience is about tipping the balance in the other direction
- But without too much logic doubt

Challenges (4)

- We want resilience wrt. timing
- One aspect is to develop methods for efficient formal verification of complex synchronization and time-triggered algorithms
    - e.g., IEEE 802 AVB, or 1588 itself
- Another is to develop algorithms and architectures for timing that are resilient wrt. system assumptions
    - e.g., do not assume an (always) synchronous substrate
    - Presumably adaptive in some way
- Finally, develop methods for formally verifying such adaptive approaches

Closing

- The New Clockwork creates opportunities through ubiquity and precision

- Some challenges are to provide extreme reliability and strong assurance
  - Former may require a thread of synchronous behavior
    - “Timely Computer Base”
  - Latter requires new(?) system models for asynchronous but synchronized computation

- Harbinger of new interest in resilience
  - Weaker (but more credible) assumptions
  - Systems that are more intricate (harder to verify)