Making Order in the Chaos: Self-stabilizing Byzantine Synchronization

Danny Dolev
The Hebrew University of Jerusalem

Joint work with A. Daliot and E. Hoch

- The focus of FuDiCo III are distributed systems that span multiple administrative domains (MADs). The workshop brings together a diverse group of computer scientists (in Systems, Theory, Security, and AI) as well as economists to discuss how to model and build systems in which nodes may deviate from their specification both because they are broken (because of bugs, misconfiguration, or even malicious attacks) and because they are selfish and are intent on maximizing their own utility

MAD-ness in the design

- Most applications assume some initial consistent state among non-faulty (honest) nodes (players)
- Most applications assume a rather simplistic fault model, if at all
- MAD systems are too complicated to reset manually
- Robustness requires considering worse case scenarios – (if cost allows)

It is not "IF" but rather "WHEN" and "TO WHAT DEGREE"

- Transient faults occur frequently in complex software systems
  - Eisenbugs, soft-bugs, difficult-to-reproduce race conditions (multi-core will introduce a new wave)
- Permanent faults add another dimension to the challenge
- Both are more frequent locally
  - Confine locally
  - Be ready to global incidents

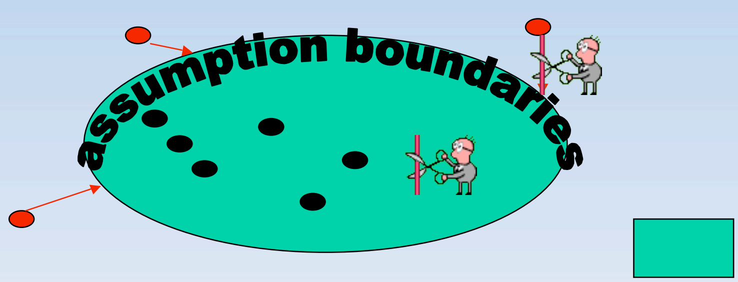
Can one tolerate transient and permanent faults at once?

Talk outline

- models

The Fault Model

Stable System:
•There exists $\boldsymbol{(n+f)/2}$ correct nodes that obey the protocol ($\boldsymbol{n>3f}$)
•No assumption about the type of behavior of some $\boldsymbol{f}$ faulty nodes
•(general network with enough connectivity)



Initial state:
arbitrary – variables at a **HONEST** node may hold arbitrary values
The code at HONEST nodes is intact

Quality of a desired solution

- Complexity as a function of the actual perturbation of the system; thus, as a function of:
  - Actual number of permanent faults
  - Type of permanent faults
  - Fraction of the system that is in an inconsistent state
  - Actual time it takes honest nodes to communicate
  - Local confinement of faults
- In addition:
  - Fast convergence
  - Low overhead
  - Applicability to general applications

Self-Stabilizing Byzantine Clock Synchronization

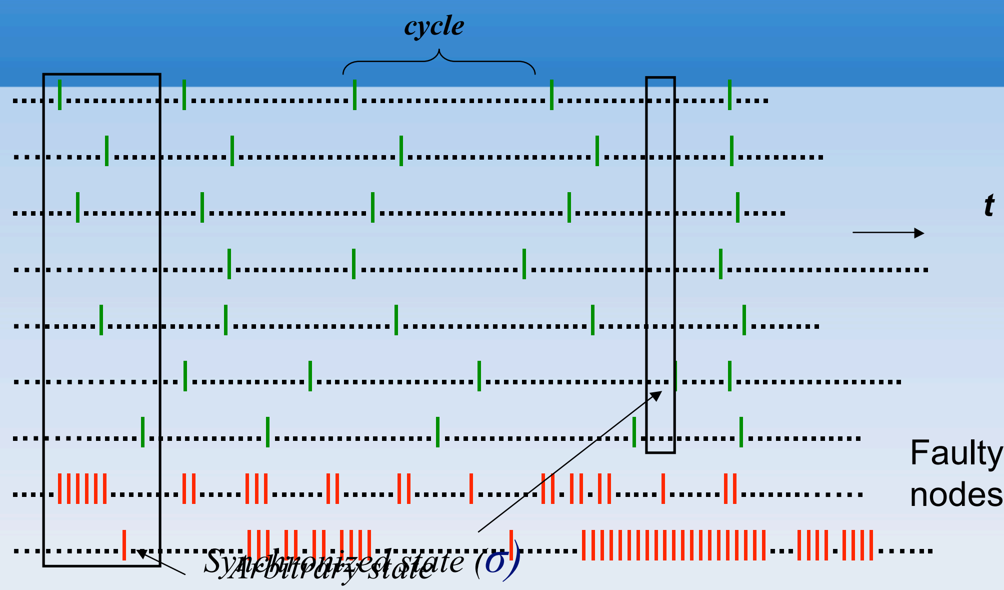
- Previously known best **self-stabilizing Byzantine** clock synchronization algorithm converges in expected $(n-f)\cdot n^{6(n-f)}$ time (S. *Dolev and J. Welch, 1995, 1997, 2004*)
- The difficulty resides in the fact that:
  - the initial clock values can differ arbitrarily
  - there is no agreed time for exchanging the values and setting the clock according to the values received
  - **clocks may wrap around**
  - Faulty nodes can try to rush the clocks out of any relation to real-time rate
  - … … …

Current state of the art

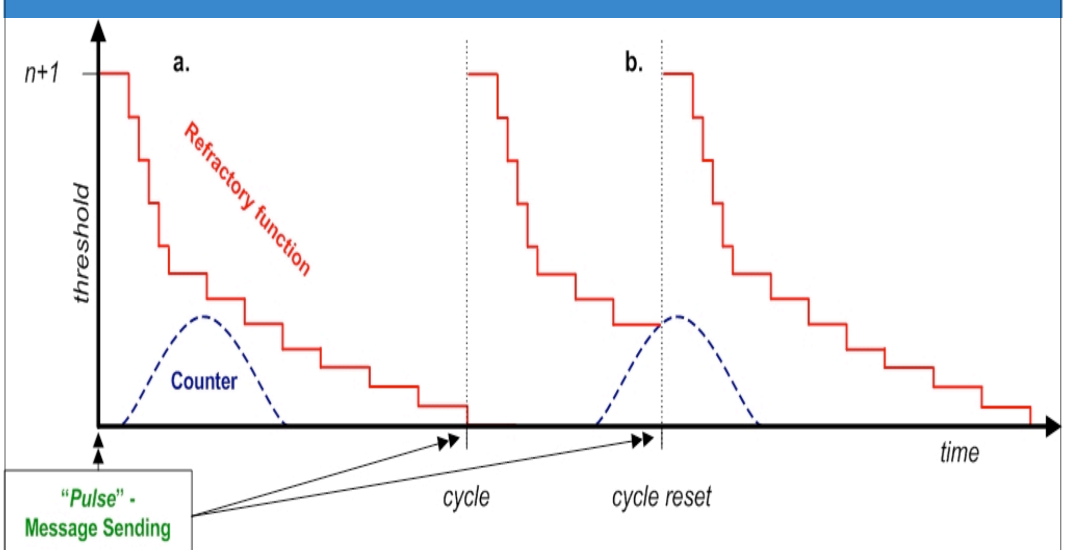
- Pulse synchronization and clock synchronization
  - Convergence -- linear time O(***f***)
  - Overhead -- a function of O(***nf+poly(f)***)
  - If only a fraction of the system is disturbed the system introduces no instability and the inconsistent parts converge fast
- These solutions can drive general translation of (many) applications that tolerate malicious faults to become also self-stabilized
- It is not a madness any more - MAD systems can be designed hierarchically with larger ***f*** locally and smaller ***f*** globally

The Real Stuff

The target is to *synchronize* pulses from *any state* and overcoming *any fault*



"*Pulse Synchronization*" algorithm



To get a sense of a solution we will present the elements of the solution in one of the possible (simple) models

Problem Statement – simple model

- two equivalent definitions:
  - Agree on special beats, spaced *Cycle beats apart*
  - *Pulse every Cycle beats*

$$[\ldots, \underbrace{1, 0, 0, \ldots, 0}_{Cycle \text{ beats}}, \underbrace{1, 0, 0, \ldots, 0}_{Cycle \text{ beats}}, \ldots]$$

- *Pulse vs. Beat*
  - *Beat comes from the global beat system*
  - *Pulse is the output of the protocol*

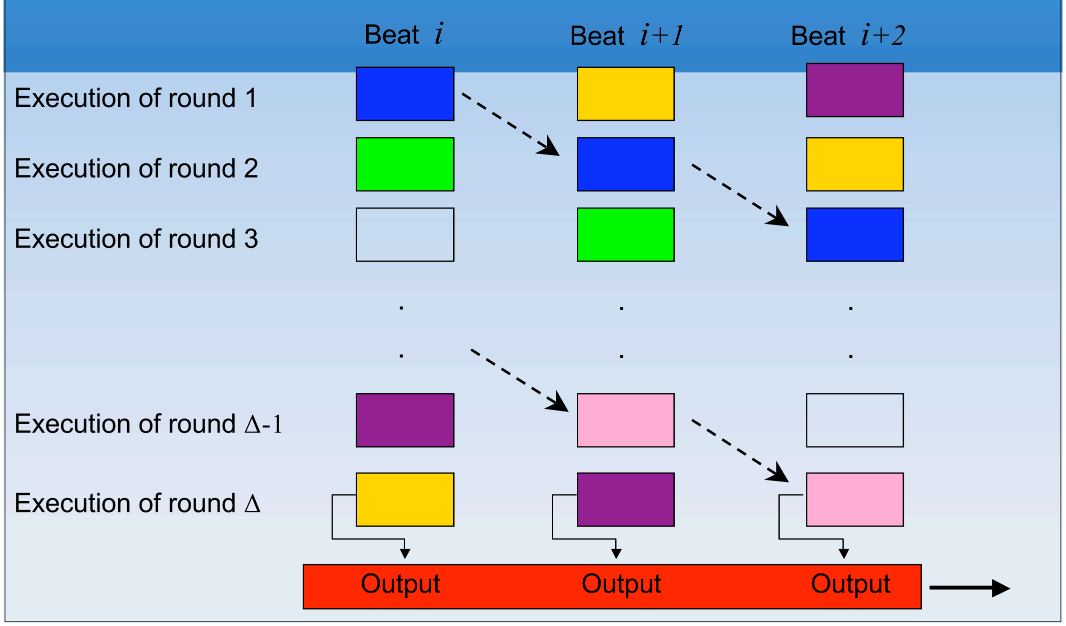
Simple Model

- ***n*** nodes
- Repetitive Global event ("beat" system)
- Network connectivity – for now assume full connectivity
- Self-stabilizing
- *Byzantine* tolerant (***f<n/3***)

Stage I – **Agreed Stream**

- **"Rotating Consensus"**: Execute simultaneously Δ *Byzantine Consensus* instances, differing at their round of execution.
  - At each beat:
    - Execute current round of each of the Δ instances
    - Output the value of the last terminated instance
    - Invoke a new instance of *Byzantine consensus*

Stage I – Contd.



Intermediate Solutions

- Stage I: $[\ddot{o}, ø]$ *-Pulser*

$$[\ldots,\underbrace{\overbrace{1,1,\ldots,1}^{\phi\text{ beats}},\overbrace{0,0,\ldots,0}^{\psi\text{ beats}}}_{Cycle\text{ beats}},\underbrace{\overbrace{1,1,\ldots,1}^{\phi\text{ beats}},\overbrace{0,0,\ldots,0}^{\psi\text{ beats}}}_{Cycle\text{ beats}},\ldots]$$

- Stage II: [Cycle]-*Pulser*

$$[\ldots,\underbrace{1,0,0,\ldots,0}_{Cycle\text{ beats}},\underbrace{1,0,0,\ldots,0}_{Cycle\text{ beats}},\ldots]$$

- Stage III: *Generalize to any Cycle value*

Quality of the solution

- Convergence is linear in *Δ, and in Cycle.*
- *For clock synchronization, Cycle is in the order of Δ, and hence the convergence is in* $O(\Delta) = O(f)$.
- *If the network is not fully connected, then* $\Delta = f \cdot D$ *where D is the diameter of the network graph. In this case, convergence is achieved in* $O(\Delta) = O(D \cdot f)$.

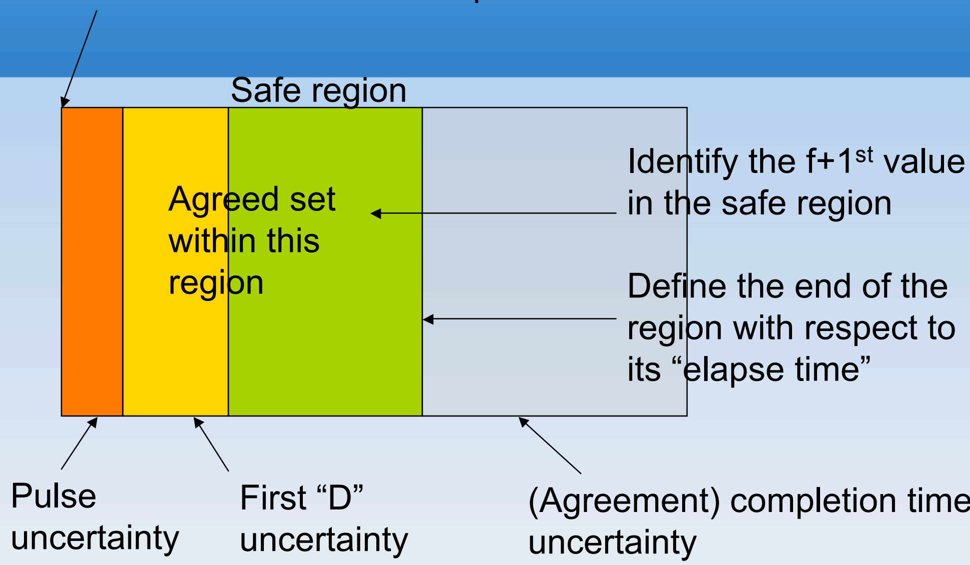
The General Scheme

Following a pulse:

- When reaching an identified state, exchange the “state” and the elapse time since the pulse
- Agree on the “state and time” sent by each node
- Collect agreed values and implicitly agree on which values to consider
- Sift the values to look for a cluster of values within D of each other or decide on a reset.
- If reset, invoke a “reset pulse”

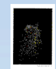
Agreed set of values

Different nodes invoke the pulse at different times



“It seems so easy… when everything is in-synch”

Synchrony phenomena in biology

- The phenomenon of synchronization is displayed by many biological systems
  - **Synchronized flashing of the male *malaccae* fireflies** 
  - Oscillations of the neurons in the circadian pacemaker, determining the day-night rhythm
  - Crickets that chirp in unison
  - Coordinated mass spawning in corals
  - Audience clapping together after a “good” performance

Synchrony phenomena in biology

- The phenomenon of synchronization is displayed by many biological systems
  - Synchronized flashing of the male *malaccae* fireflies
  - **Oscillations of the neurons in the circadian pacemaker, determining the day-night rhythm**
  - Crickets that chirp in unison
  - Coordinated mass spawning in corals
  - Audience clapping together after a “good” performance

Synchrony phenomena in biology

- The phenomenon of synchronization is displayed by many biological systems
  - Synchronized flashing of the male *malaccae* fireflies
  - Oscillations of the neurons in the circadian pacemaker, determining the day-night rhythm
  - **Crickets that chirp in unison**
  - Coordinated mass spawning in corals
  - Audience clapping together after a "good" performance

Synchrony phenomena in biology

- The phenomenon of synchronization is displayed by many biological systems
  - Synchronized flashing of the male *malaccae* fireflies
  - Oscillations of the neurons in the circadian pacemaker, determining the day-night rhythm
  - Crickets that chirp in unison
  - **Coordinated mass spawning in corals**

  - Audience clapping together after a "good" performance

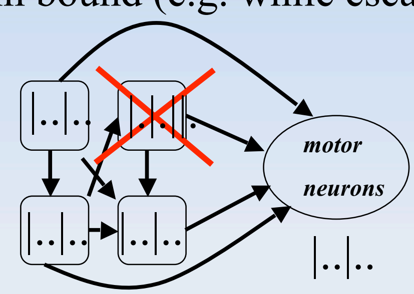
Synchrony phenomena in biology

- The phenomenon of synchronization is displayed by many biological systems
  - Synchronized flashing of the male *malaccae* fireflies
  - Oscillations of the neurons in the circadian pacemaker, determining the day-night rhythm
  - Crickets that chirp in unison
  - Coordinated mass spawning in corals
  - **Audience clapping together after a "good" performance**

Cardiac ganglion of the lobster

(*Sivan, Dolev & Parnas, 2000*)

- Four interneurons **tightly synchronize** their pulses in order to give the heart its optimal pulse rate (though one is enough for activation)
- Able to **adjust** the **synchronized** firing pace, up to a certain bound (e.g. while escaping a predator)



Cardiac ganglion of the lobster
(*Sivan, Dolev & Parnas, 2000*)

- Must not fire out of synchrony for prolonged times in spite of
  - **Noise**
  - Single **neuron death**
  - Inherent **variations** in the firing rate
  - Firing frequency regulating **Neurohormones**
  - **Temperature** changes

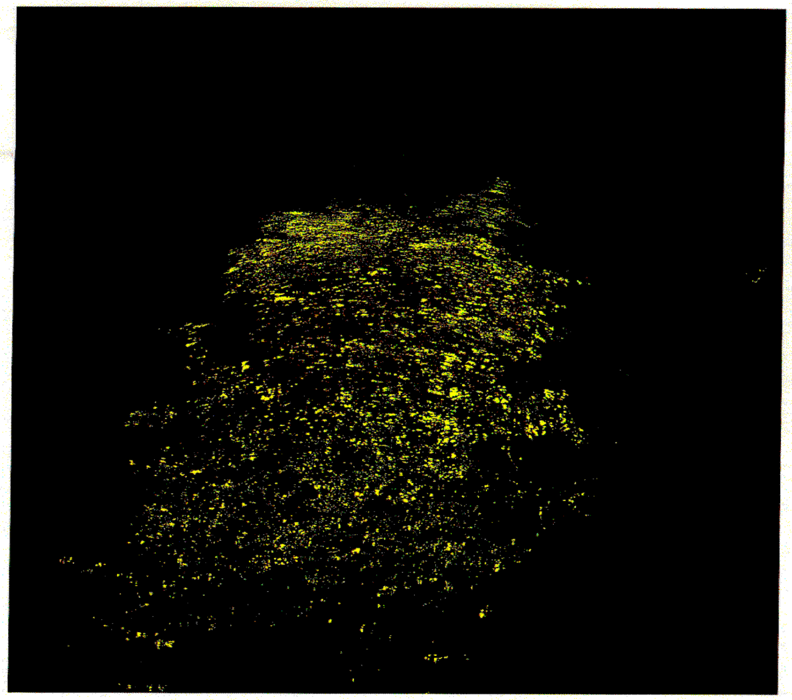
- The vitality of the cardiac ganglion suggests it has evolved to be optimized for
  - Fault tolerance
  - Re-synchronization from any state (“self-stabilization”)
  - Tight synchronization
  - Fast re-synchronization

Questions?

Proposed approach

- Design the system to establish locality and prevent instability resulting from a single or few unstable elements
- Establish time reference (if no outside one exists)
- Produce an agreed-upon event in the flow of events
- Assign to it an agreed value
- Use it to anchor the application detection mechanism
- Add blocking mechanism
- Add correcting mechanism
- Repeatedly invoke the mechanisms in the background

Thank you!!!



SCIENTIFIC AMERICAN *December 1993* 69