A Byzantine-Fault Tolerant Self-Stabilizing Protocol for Distributed Clock Synchronization Systems

Mahyar R. Malekpour
NASA-Langley Research Center
m.r.malekpour@larc.nasa.gov
+1 757-864-1513
http://shemesh.larc.nasa.gov/people/mahyar.htm

Why Stabilization?

- Initialization
- Recovery from random, independent, transient failures
- Recovery from massive correlated failures

What is the Stabilization of Clock Synchronization Problem?

- In electrical engineering terms, for digital logic and data transfer, a synchronous object requires a clock signal.
- A distributed synchronous system requires a logical clock signal.
- Synchronization means coordination of simultaneous threads or processes to complete a task in order to get correct runtime order and avoid unexpected race conditions.

- Stabilization of clock synchronization is bringing the logical clocks of a distributed system *in sync* with each other (hence, title of this report).

How to Achieve Stabilization?

- External Control (centralized, master-target)
  - Direct
    - Power on/Cold Reset
    - Hot Reset
    - Master switch
  - Indirect
    - GPS, i.e. time (synchronous)
    - Go/Start command (asynchronous)

Great for close proximity

- Problems
  - GPS is not always reliable
  - There is no GPS on Mars
  - Central command is impractical over long distances

How to Achieve Stabilization?

- Internal Control (distributed)
  - Local awareness about self and state of the system (diagnosis)
  - Coordination with others (synchrony)
  - Cooperation with others (agreement)

Self-Stabilization

- Problems
  - Awareness
  - Establish synchrony
  - Establish agreement
    - On critical states; schedule, membership
  - Maintain synchrony
  - Maintain agreement

Diagnosis

Convergence

Closure

Byzantine General Problem

- Leslie Lamport, Marshall Pease and Robert Shostak
  - Distributed computing and Chinese Generals Problem
    - Two generals need to agree on attack or retreat
    - Communicate via sending messengers who might never arrive
  - The Byzantine Generals Problem, published in 1982
    - Generalization of the Chinese General Problem
- Dismissed as a theoretical problem, based on low probability
- Kevin Driscoll, et al, 2003, “Byzantine Fault Tolerance, from Theory to Reality”
  - Probability of asymmetric faults is not as low as it is usually assumed to be.
  - A system with high reliability requirements has to be designed to handle such faults.

What is known?

- Agreement can be guaranteed only if $K \geq 3F + 1$,
  - K is the total number of nodes and F is the maximum number of faulty nodes.
  - E.g. need at least 4 nodes just to tolerate 1 fault.
- Re-synchronization cycle or period, P, to prevent too much deviation in clocks/timers.
- There are many partial solutions based on strong assumptions (initial synchrony, or existence of a common pulse).
- There are clock synchronization algorithms that are based on randomization and are non-deterministic.
- There are claims that cannot be substantiated.
- There is no guideline for how to solve this problem or documented pitfalls to avoid in the process.
- Speculation on proof of impossibility.
- There was no solution for the general case.

Why is this problem difficult to solve?

- This problem is hard to solve and just as hard to prove.

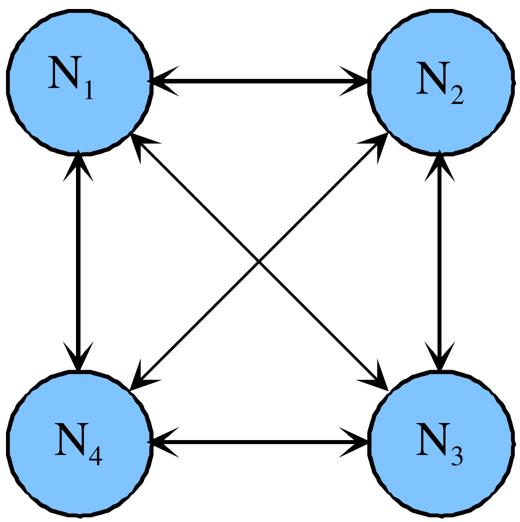
- Aspects of Complexity
  – Design of a solution
  – Composition of a paper-and-pencil proof of the solution
  – Validation of the paper-and-pencil proof
  – Mechanical proof of the solution

The Approach

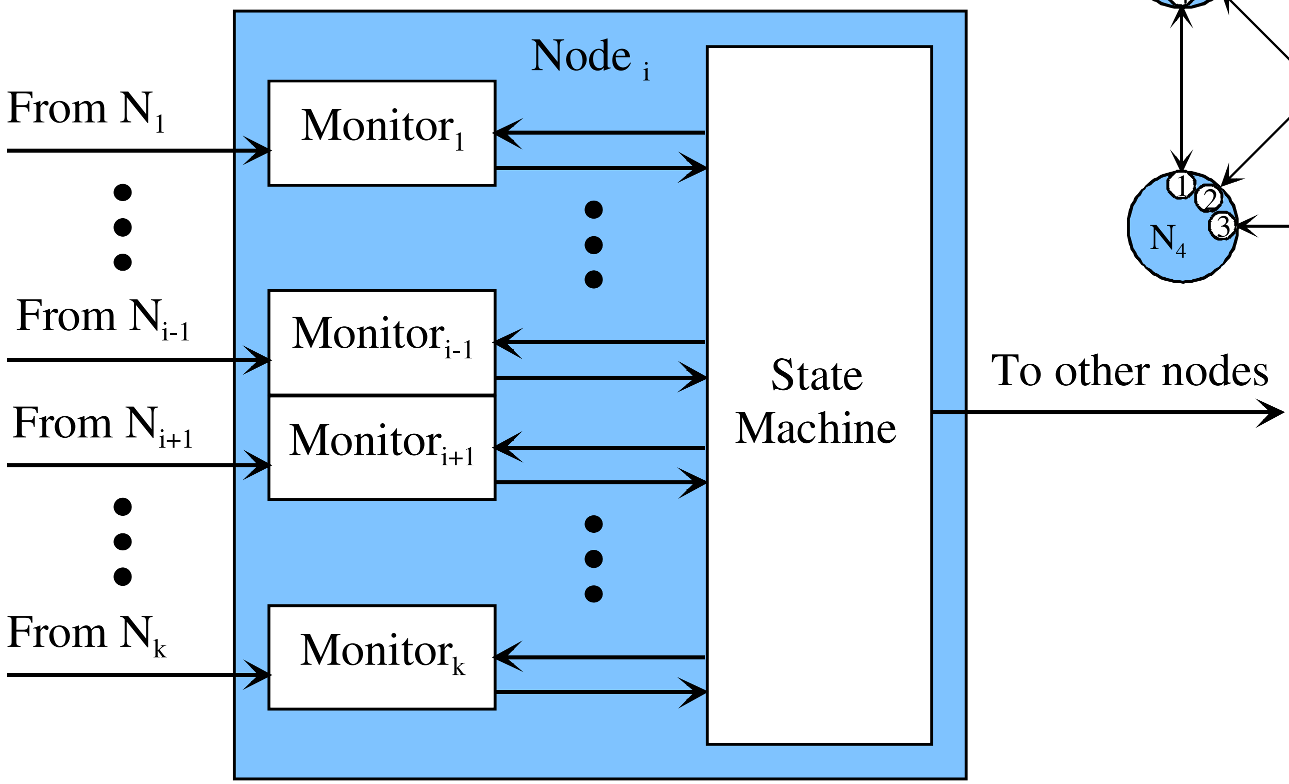
- The approach is dynamic and gradual.
  - It takes time; convergence is not spontaneous
  - Requires continuous vigilance and participation
  - Based on system awareness (feedback), i.e. local diagnosis
  - Understanding the relationship between time and event
- It is a feedback control system.

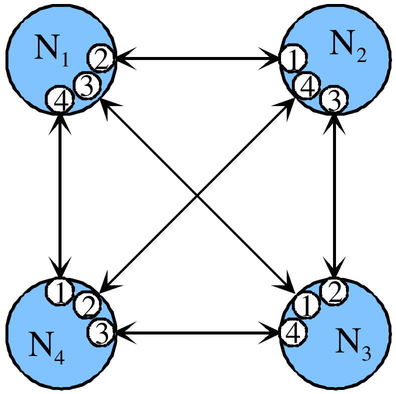
Topology

- The source of a message is distinctly identifiable by the receivers from other sources of messages.
- E.g. a fully connected graph.

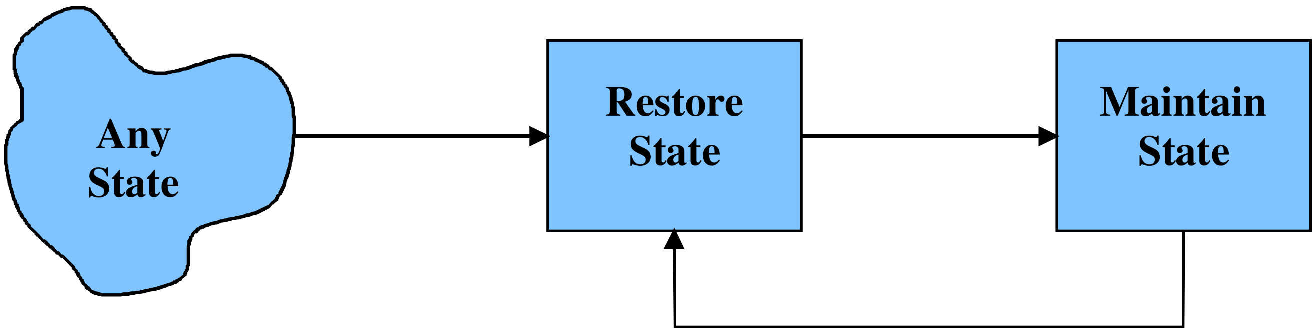


Nodes and Monitors





The Idea



- Bring all good nodes to the *Restore* state.
  - Asynchronous process
- Transition all good nodes from *Restore* state to *Maintain* state.
  - Synchronous process
  - Within a guaranteed initial precision
- Maintain bounded synchrony by repeating this process periodically.

State Transitions

- Transition from *Maintain* state to *Restore* state:
  - *Retry()* or *TimeOutMaintain()*
    - At least one good node in *Restore* state or time to resync.
- Transition from *Restore* state to *Maintain* state:
  - Based on the **transitory conditions**
    - The node is in the *Restore* state,
    - At least $2F$ *Accept()* in as many Δ_{AA} intervals after the node entered the *Restore* state,
    - No *valid Resync* messages are received for the last *Accept()*.
- Duration of the *transitory delay* (during the steady state) is bounded by $[2F, 3F]$.

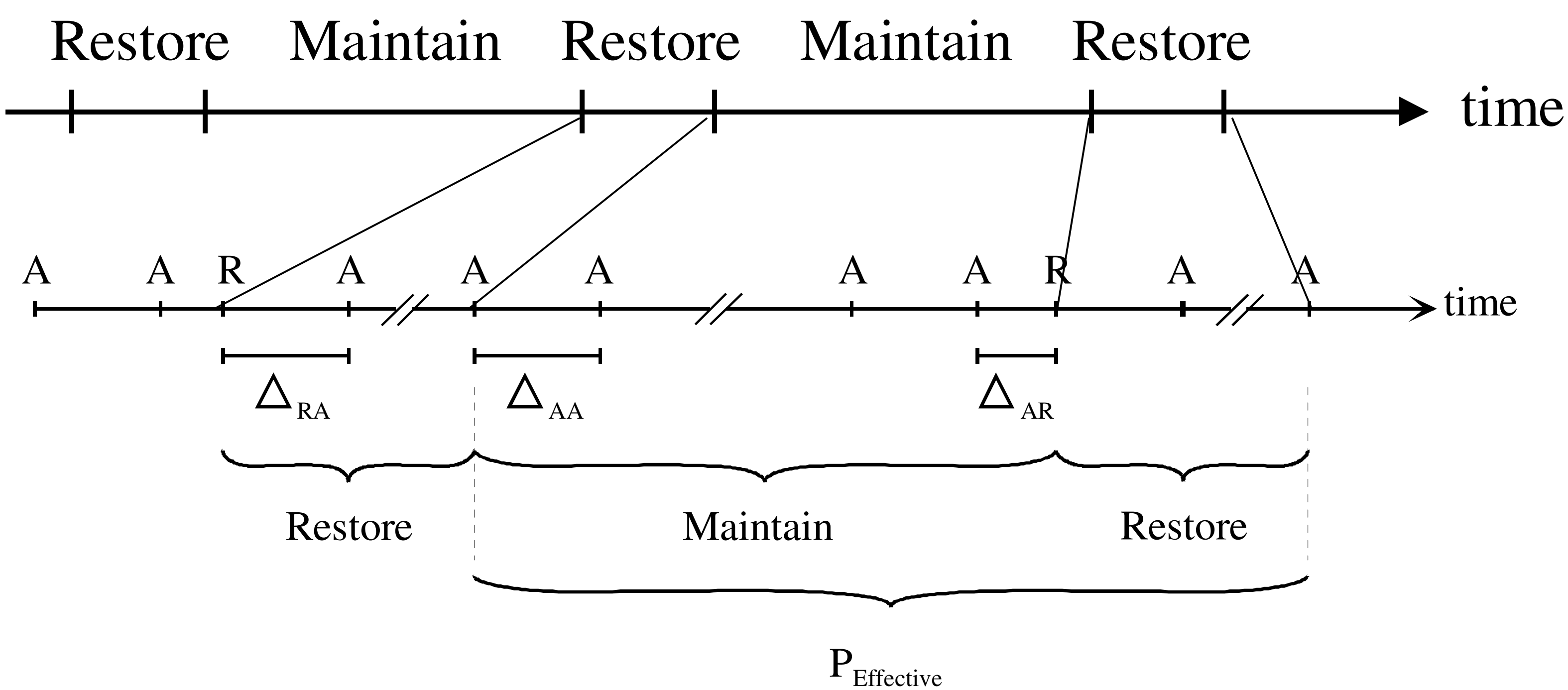
Messages

- Protocol messages: *Resync* and *Affirm*
- *Resync*, *R* for short, is sent when *Retry(), TimeOutRestore(),* or *TimeOutMaintain()*.
- *Affirm*, *A* for short, is sent at Δ_{AA} intervals when *TimeOutAcceptEvent()*.
  - Sent periodically to reduce error detection time, expedite convergence, and achieve tighter precision.
- A good node does not use its own message.

Timers

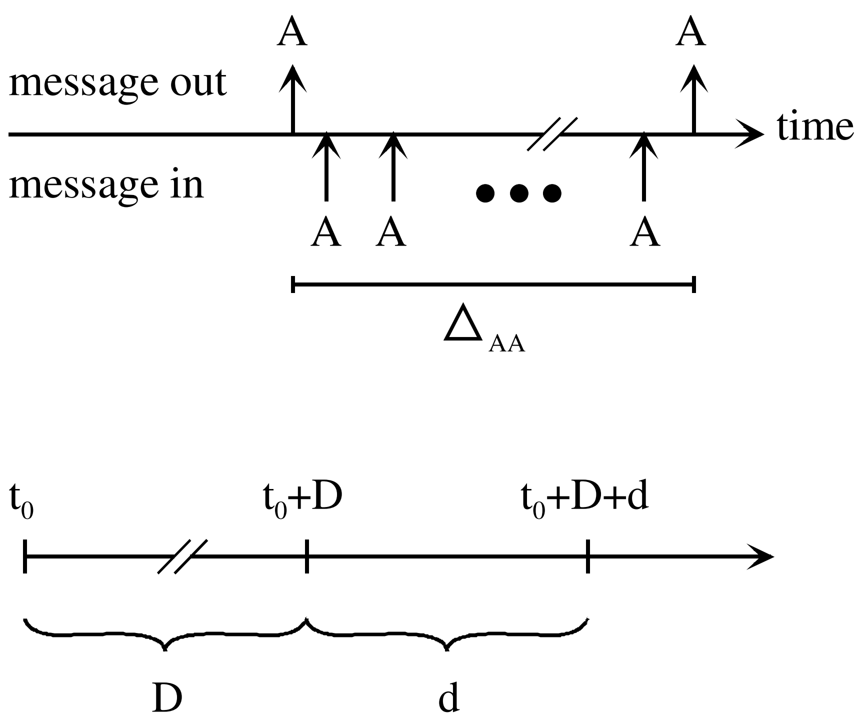
- A node keeps track of two logical timers:
  - *State_Timer*, reflects the duration of the current state.
    - Reset whenever entering a state (*Restore* or *Maintain*).
  - *Local_Timer*, used in assessing the state of the system.
    - Reset in the *Maintain* state when *State_Timer* $= \lceil \Delta_{Precision} \rceil$.
- These timers are incremented once per Δ_{AA}.
- *Restore* state, *T* for short, maximum duration is P_T.
- *Maintain* state, *M* for short, maximum duration is $P_M \geq P_T$.

Steady State System Behavior



- Expected message sequence:
  - *RAAA … AAAR*

Determining Δ's



- Min event-response delay, $D \geq 1$
- Network imprecision, $d \geq 0$
- $\Delta_{AA} \geq (D + d)$
- $\Delta_{RA} = \Delta_{AA}$
- $1 \leq \Delta_{AR} \leq \Delta_{AA}$
- $\Delta_{Precision} = (3F - 1)\ \Delta_{AA} - D + \Delta_{Drift}$
- $\Delta_{Drift} = ((1+\rho) - 1/(1+\rho))\ P_M\ \Delta_{AA}$

The Protocol - Monitor

case (incoming message from the corresponding node)
{*Resync*:

if *InvalidResync()* then

Invalidate the message

else

Validate and store the message,

Set state status of the source.

***Affirm*:**

if *InvalidAffirm()* then

Invalidate the message

else

Validate and store the message.

***Other*:**

Do nothing.

} // **case**

The Protocol - Node

case (state of the node)
{*Restore*:
if *TimeOutRestore()* then
Transmit *Resync* message,
Reset *State_Timer*,
Reset *DeltaAA_Timer*,
Reset *Accept_Event_Counter*,
Stay in *Restore* state,

elsif *TimeOutAcceptEvent()* then
Transmit *Affirm* message,
Reset *DeltaAA_Timer*,
if *Accept()* then
Consume *valid* messages,
Clear state status of the sources,
Increment *Accept_Event_Counter*,
if *TransitoryConditionsMet()* then
Reset *State_Timer*,
Go to *Maintain* state,
else
Stay in *Restore* state.
else
Stay in *Restore* state.,
else
Stay in *Restore* state.

***Maintain*:**
if *TimeOutMaintain()* or *Retry()* then
Transmit *Resync* message,
Reset *State_Timer*,
Reset *DeltaAA_Timer*,
Reset *Accept_Event_Counter*,
Go to *Restore* state,

elsif *TimeOutAcceptEvent()* then
if *Accept()* then
Consume *valid* messages.,
if ($State\_Timer = \lceil \Delta_{Precision} \rceil$)
Reset *Local_Timer*.,
Transmit *Affirm* message,
Reset *DeltaAA_Timer*,
Stay in *Maintain* state,

else
Stay in *Maintain* state.
} // **case**

Paper-and-pencil proof

System Assumptions

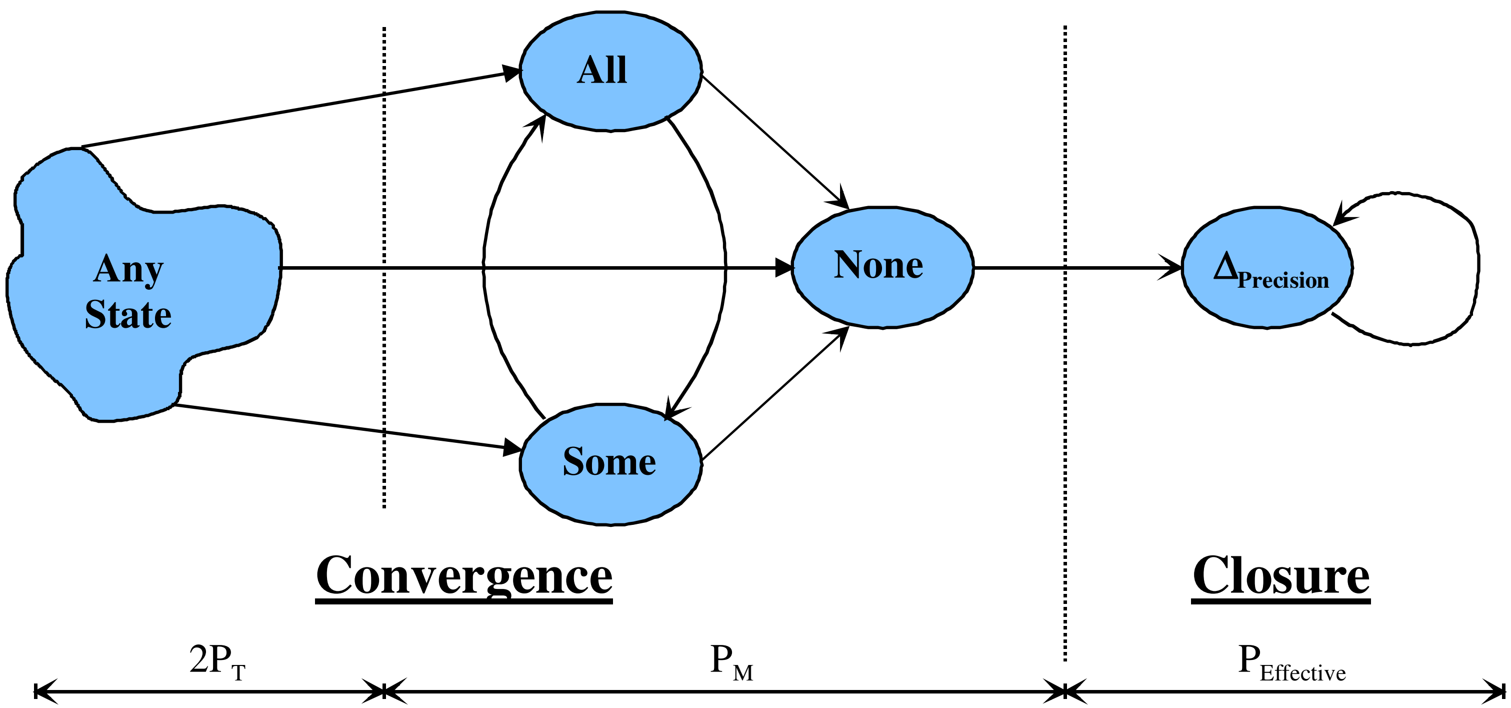
- The cause of the transient faults (disturbance) has dissipated.
- All good nodes actively participate in the self-stabilization process and execute the protocol.
- At most F of the nodes are faulty.
- The source of a message is distinctly identifiable by the receivers from other sources of messages.
- A message sent by a good node will be received and processed by all other good nodes within Δ_{AA}, where $\Delta_{AA} \geq (D + d)$.
- The initial values of the state and all variables of a node can be set to any arbitrary value within their corresponding range.

Properties of the Protocol

- **Convergence** - From any state, the system converges to a self-stabilized state after a finite amount of time.
  - $\forall\ N_i, N_j \in K_G$, $\Delta_{Local\_Timer}(C) \leq \Delta_{Precision}$.
  - $\forall\ N_i, N_j \in K_G$, at C, N_i perceives N_j as being in the *Maintain* state.

- **Closure** - When all good nodes have converged to a given self-stabilization precision, $\Delta_{Precision}$, at time C, the system shall remain within the self-stabilization precision $\Delta_{Precision}$ for $t \geq C$, for real time t.
  - $\forall\ N_i, N_j \in K_G$, $t \geq C$, $\Delta_{Local\_Timer}(t) \leq \Delta_{Precision}$,

Proof - Approach



- **All good nodes are in the *Maintain* state.**
- **Some of the good nodes are in the *Maintain* state.**
- **None of the good nodes are in the *Maintain* state.**

Proof - Sketch

Theorem StabilizeFromAnyState – *A system of $K \geq 3F + 1$ nodes self-stabilizes from any random state after a finite amount of time.*

- **Theorem *ResyncWithinP$_T$*, Theorem *RestoreToMaintain*, and Corollary *RestoreToMaintainWithin2P$_T$* –**
  - 1- **None** of the good nodes are in the Maintain state
  - 2- **All** good nodes are in the Maintain state
  - 3- **Some** of the good nodes are in the Maintain state
- **Convergence – *None of the good nodes are in the Maintain state*:**
  It follows from Theorems *ConvergeNoneMaintain* and *ClosureAllMaintain* that such system always self-stabilizes.
- **Convergence – *All good nodes are in the Maintain state*:**
  It follows from Theorems *ConvergeNoneMaintain*, *ConvergeAllMaintain* and *ClosureAllMaintain* that such system always self-stabilizes.
- **Convergence – *Some of the good nodes are in the Maintain state*:**
  It follows from Theorems *ConvergeNoneMaintain*, *ConvergeAllMaintain*, *ConvergeSomeMaintain*, and *ClosureAllMaintain* that such system always self-stabilizes.

Proof - Sketch

- **Mutually Stabilized** – *$\forall N_i, N_j \in K_G$, at C, N_i perceives N_j as being in the Maintain state.*

  It follows from Corollary *MutuallyStabilized* that all good nodes mutually perceive each other to be in the *Maintain* state.

- **Closure:** *When all good nodes have converged such that $\Delta_{Local\_Timer}(C) \leq \Delta_{Precision}$, at time C, the system shall remain within the self-stabilization precision $\Delta_{Precision}$ for $t \geq C$, for real time t.*

  It follows from Theorems *ClosureAllMaintain* and *LocalTimerWithinPrecision* that such system always remains stabilized and $\Delta_{Local\_Timer}(t) \leq \Delta_{Precision}$ for $t \geq C$. ♦

Proof via Model Checking

- Main problem, state space explosion
- Used SMV and successfully model checked the protocol for the **base case** (deceptively simple):
  - Fully connected graph
  - 4-node system, 3 good nodes, 1 Byzantine faulty node
  - $D = 1$, $d = 0$, $\Delta_{AA} = 1$, $\rho = 0$
  - Initially 4.26×10^{46} states
  - After abstraction and reduction techniques, 5.13×10^{24} states
  - PC, running Linux, 4GB memory
- Model checking effort took over two years
  - Report under review

Protocol Characteristics

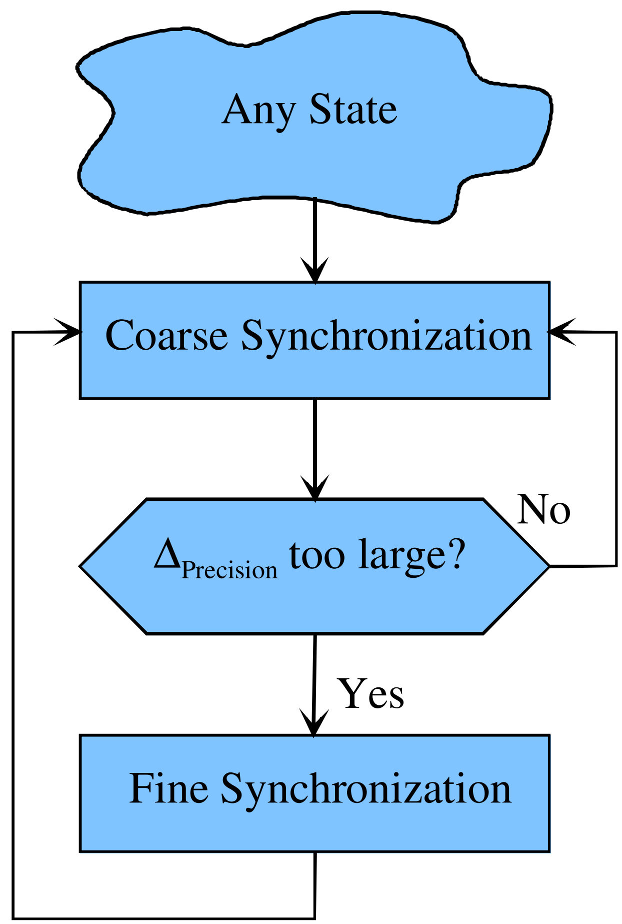
- Self-stabilizes in the presence of permanent Byzantine failures.
  - From any initial random state
  - Tolerates bursts of random, independent, transient failures
  - Recovers from massive correlated failures
- Convergence
  - Deterministic
  - Bounded
  - Linear-time with respect to the self-stabilization period, P_M.
- Rapid; converges within one self-stabilization period, P_M.
- Low overhead, $1/w$, w is the width of the data message.
- All timing measures of variables are based on the node's local clock.
- Scalable with respect to the *fundamental parameters* K, D and d.
- No central clock or externally generated pulse is used.
- Does not require global diagnosis, but $K \geq 3F + 1$.

Achieving Tighter Precision

- If Δ_{AA}, and hence $\Delta_{Precision}$, is larger than the desired precision, the system is said to be **Coarsely Synchronized.** Otherwise, the system is said to be **Finely Synchronized**.
- The desired precision can be achieved in a two step process.
  - First, the system has to be *Coarsely Synchronized* and guaranteed that the system remains *Coarsely Synchronized* and operates within a known precision, $\Delta_{Precision}$.
  - Second, utilize a proven protocol that is based on the initial synchrony assumptions to achieve optimum precision.
    - E.g. Fault-Tolerant Mid-Point function (FTMP) or Fault-Tolerant Averaging function (FTA), FTMP = floor ((T_{F+1} + T_{K-F}) / 2).

- Topic of my next report.

The interplay of *Coarsely* and *Finely Synchronized* protocols.



Future Plans

- Build it and show that it works in a harsh environment, e.g. High Intensity Radiated Field (HIRF), neutron radiation.
  - 4-node system (*base case*)
  - Have capability for up to 14-node system
- Integration of this protocol with a *Finely Synchronized* protocol.
- Adapting to SPIDER topology
- Adapting to other topologies
- Hybrid fault models
- Dynamic node count
- Continue model checking of larger and more complex systems.

Questions?