A Self-Stabilizing Byzantine-Fault-Tolerant Clock Synchronization Protocol

Mahyar R. Malekpour
Langley Research Center, Hampton, Virginia

June 2009
Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA scientific and technical information (STI) program plays a key part in helping NASA maintain this important role.

The NASA STI program operates under the auspices of the Agency Chief Information Officer. It collects, organizes, provides for archiving, and disseminates NASA's STI. The NASA STI program provides access to the NASA Aeronautics and Space Database and its public interface, the NASA Technical Report Server, thus providing one of the largest collections of aeronautical and space science STI in the world. Results are published in both non-NASA channels and by NASA in the NASA STI Report Series, which includes the following report types:

- **TECHNICAL PUBLICATION.** Reports of completed research or a major significant phase of research that present the results of NASA programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA counterpart of peer-reviewed formal professional papers, but having less stringent limitations on manuscript length and extent of graphic presentations.

- **TECHNICAL MEMORANDUM.** Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.

- **CONTRACTOR REPORT.** Scientific and technical findings by NASA-sponsored contractors and grantees.

- **CONFERENCE PUBLICATION.** Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or co-sponsored by NASA.

- **SPECIAL PUBLICATION.** Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.

- **TECHNICAL TRANSLATION.** English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services also include creating custom thesauri, building customized databases, and organizing and publishing research results.

For more information about the NASA STI program, see the following:

- Access the NASA STI program home page at [http://www.sti.nasa.gov](http://www.sti.nasa.gov)

- E-mail your question via the Internet to help@sti.nasa.gov

- Fax your question to the NASA STI Help Desk at 443-757-5803

- Phone the NASA STI Help Desk at 443-757-5802

- Write to:
  NASA STI Help Desk
  NASA Center for AeroSpace Information
  7115 Standard Drive
  Hanover, MD 21076-1320
A Self-Stabilizing Byzantine-Fault-Tolerant Clock Synchronization Protocol

Mahyar R. Malekpour
Langley Research Center, Hampton, Virginia
Acknowledgments

This effort was conducted under the Integrated Vehicle Health Management project of NASA’s Aviation Safety program. The author would like to thank the following for their reviews, helpful comments, consultations and support: Ricky Butler, Victor Carreno, Eric Cooper, Jeff Maddalon, Ben DiVito, Paul Miner, Cesar Munoz, Radu Siminiceanu, and Kristin Rozier. The author would like to especially thank Wilfredo Torres-Pomales for his in-depth reviews. The author would also like to thank Celeste Belcastro without whose support this work would not have been possible.
Abstract

This report presents a rapid Byzantine-fault-tolerant self-stabilizing clock synchronization protocol that is independent of application-specific requirements. It is focused on clock synchronization of a system in the presence of Byzantine faults after the cause of any transient faults has dissipated. A model of this protocol is mechanically verified using the Symbolic Model Verifier (SMV) [SMV] where the entire state space is examined and proven to self-stabilize in the presence of one arbitrary faulty node. Instances of the protocol are proven to tolerate bursts of transient failures and deterministically converge with a linear time with respect to the synchronization period. This protocol does not rely on assumptions about the initial state of the system, other than the presence of sufficient number of good nodes. All timing measures of variables are based on the node’s local clock, and no central clock or externally generated pulse is used. The Byzantine faulty behavior modeled here is a node with arbitrarily malicious behavior that is allowed to influence other nodes at every clock tick. The only constraint is that the interactions are restricted to defined interfaces.
# Table of Contents

1. **INTRODUCTION** ........................................................................................................................................... 1

2. **SYSTEM OVERVIEW** ................................................................................................................................. 2
   2.1. GAMMA (γ) .............................................................................................................................................. 3

3. **PROTOCOL DESCRIPTION** ......................................................................................................................... 3
   3.2. THE MONITOR ......................................................................................................................................... 4
   3.3. THE STATE MACHINE ............................................................................................................................... 4
   3.4. PROTOCOL FUNCTIONS ......................................................................................................................... 7
   3.5. PROTOCOL ASSUMPTIONS ...................................................................................................................... 8
   3.6. THE SELF-STABILIZING CLOCK SYNCHRONIZATION PROBLEM ....................................................... 8

4. **THE CLOCK SYNCHRONIZATION PROTOCOL** ............................................................................................ 10

5. **PROOF** ....................................................................................................................................................... 11
   5.1. PROOF FOR F = 1................................................................................................................................. 22
      5.1.1. In-Phase Case ............................................................................................................................... 23
      5.1.2. Out-of-Phase Case ....................................................................................................................... 26
      5.1.3. A Realizable System ..................................................................................................................... 27
   5.2. PROOF FOR F = 0................................................................................................................................. 29
   5.3. GENERALIZATION OF THE PROTOCOL, FOR F > 1 ........................................................................... 31

6. **PROTOCOL OVERHEAD** ............................................................................................................................ 32

7. **POSSIBLE APPLICATIONS** .......................................................................................................................... 33

8. **CONCLUSIONS** ......................................................................................................................................... 33

REFERENCES ....................................................................................................................................................... 35

APPENDIX A. **SYMBOLS** .................................................................................................................................. 36
1. Introduction

This report presents a clock synchronization protocol and the proof of its correctness for specific cases. For an introduction to the clock synchronization and self-stabilization problems the reader is referred to the introductory sections in [Mal 2006A, 2006B, 2007, 2008].

A Byzantine-Fault-Tolerant Self-Stabilizing Protocol for Distributed Clock Synchronization Systems was reported in [Mal 2006A, 2006B, 2007, 2008]. Claims about the protocol were validated via mechanical verification of a system consisting of one Byzantine faulty node [Mal 2007, 2008]. Further analysis of the proofs revealed a potential simplification of this protocol. Having mechanically verified the protocol, it is now possible to explore variations of the protocol. What is presented here is a new protocol that is a direct result of this exploration and re-verification of the protocol reported in [Mal 2006A, 2006B, 2007, 2008].

The protocol in [Mal 2006A, 2006B, 2007] requires periodic transmission of Affirm messages to guarantee the presence and participation of all good nodes. Assuming that the good nodes are actively participating in the self-stabilization process, periodic transmission of Affirm messages can be inferred and, thus, periodic arrival of Affirm messages can be assumed. As a result, transmission of the Affirm messages can be eliminated. Therefore, only one self-stabilization message, Resync, suffices. Nevertheless, it is worth noting that periodic transmission of the Affirm messages by the good nodes not only reduces the error detection time but also expedites the reintegration process. In [Mal 2006A, 2006B, 2007, 2008] an accept event counter was introduced to account for the arrival and accumulation of Affirm messages. Assuming the presence of periodic Affirm messages, the corresponding behavior of the protocol is now compensated by keeping track of elapsed time.

This report extends the results of the basic case [Mal 2007, 2008] to larger systems and examines synchronization of a system of $K \geq 3F + 1$ nodes in the presence of multiple Byzantine faulty nodes. Analysis of larger systems revealed that a direct generalization of the results of the basic case is not applicable to larger systems. Although this protocol solves the basic case, it does not synchronize a system of $K \geq 3F + 1$ nodes in the presence of $F$ Byzantine faulty nodes when $F > 1$.

The proof presented here applies to all instances and applications of this protocol and to those reported in [Mal 2006A, 2006B, 2007, 2008]. Although this proof parallels that of [Mal 2006A, 2006B, 2007, 2008], the proof is redone and restructured to make it easier to follow, simpler to analyze, and, thus, easier to comprehend. Elimination of the Affirm messages resulted in a reduction of the number of parameters and, hence, the initial state space. The mechanical verification of the protocol presented in this report is now more manageable and can be conducted in shorter amount of time on computers with less memory. Furthermore, if more memory and computing power were available, larger and more complex systems could be analyzed. Also, in the absence of periodic transmission of Affirm messages, implementation of the implicit fault model [Mal 2007] of the faulty nodes in the mechanical verification models is now practical.
In this report, a rapid Byzantine self-stabilizing clock synchronization protocol is presented. Specific cases of this protocol are demonstrated to self-stabilize from any state, tolerate bursts of transient failures, and deterministically converge within a linear time with respect to the synchronization period. Upon stabilization, all good clocks proceed synchronously.

2. System Overview

The underlying topology considered here is a network of \( K \geq 3F + 1 \) nodes that exchange messages through a set of communication channels. A maximum of \( F \) Byzantine faulty nodes are assumed to be present in the system, where \( F \geq 0 \). The communication channels are assumed to connect a set of source nodes to a set of destination nodes such that the source of a given message is uniquely identifiable from other sources of messages. The minimum number of good nodes in the system, \( G \), is defined by \( G = K - F \). The nodes communicate with each other by exchanging broadcast messages. Broadcast of a message to all other nodes is realized by transmitting the message to all other nodes at the same time. The communication network does not guarantee any relative order of arrival of a broadcast message at the receiving nodes, and a consistent delivery order of a set of messages does not necessarily reflect the temporal or causal order of the message transmissions [Kop 1997].

Each node is driven by an independent local physical oscillator with one oscillation representing a local clock tick. The oscillators of good nodes have a known bounded drift rate, \( 0 \leq \rho \ll 1 \), with respect to real time.

Each node has two primary logical time clocks, \( \text{StateTimer} \) and \( \text{LocalTimer} \), which locally keep track of the passage of time as indicated by the local clock tick. There is neither a central system clock nor an externally generated global pulse.

The faulty communication channels and nodes can behave arbitrarily provided that eventually the system adheres to the protocol assumptions (see Section 3.5).

The communication latency between the nodes is expressed in terms of the minimum event-response delay, \( D \), and network imprecision, \( d \). These parameters are described with the help of Figure 1. As depicted, a message transmitted at real time \( t_0 \) is expected to arrive at all destination good nodes, be processed, and subsequent messages generated within the time interval of \( [t_0 + D, t_0 + D + d] \). Communication between independently clocked nodes is inherently imprecise. The network imprecision, \( d \), is the maximum time difference among all good receivers, \( N_j \), of a message from good node \( N_i \) with respect to real time. The imprecision is due to the drift of the clocks with respect to real time, jitter, discretization error, temperature effects and differences in the lengths of the physical communication media. These two parameters are assumed to be bounded such that \( D \geq 1 \) and \( d \geq 0 \) and both have values with units of nominal clock tick.
Figure 1. Event-response delay, $D$, and network imprecision, $d$.

2.1. Gamma ($\gamma$)

The timeline of activities of a node is partitioned into a sequence of equal duration intervals from the time the node transitioned to a new state. The duration of these intervals, denoted $\gamma$, is expressed in terms of $D$ and $d$, constrained such that $\gamma \geq (D + d)$, and measured by the local oscillator. Unless stated otherwise, all time-dependent parameters of this protocol are measured locally and expressed as functions of $\gamma$. The time-driven activities of a node take place at equal intervals measured by the local oscillator since the node entered a new state. In contrast, event-driven activities are independent of $\gamma$ and, thus, take place immediately.

3. Protocol Description

The system is in the steady state when it is stabilized. In order to achieve stabilization, the nodes communicate by exchanging a Sync message. A Sync message is transmitted either as a result of a resynchronization timeout, or when a node determines that sufficient number of other nodes have engaged in the resynchronization process.

Three fundamental parameters characterize the self-stabilization protocol presented here, namely $K$, $D$, and $d$. The maximum number of faulty nodes, $F$, the minimum number of good nodes, $G$, the $\gamma$ intervals, and the remaining parameters that are subsequently presented are derived parameters based on the fundamental parameters.

3.1. Message Validity

Only one message is required for the operation of the protocol. Receiving a Sync message is indicative of its validity in the value domain. The protocol performs as intended when the timing requirements of the received messages from all good nodes at all other good nodes are satisfied. The time interval between any two consecutive Sync messages from a node is denoted $\Delta_{SS}$, and the shortest such interval for a good node is denoted $\Delta_{SS,min}$. The following definitions apply at the receiving nodes.

- A Sync message from a given source is valid if it arrives at or after $\Delta_{SS,min}$ of an immediately preceding Sync message that is valid in the value domain.
- While in the Maintain state, a Sync message from a given source remains valid for the duration of that state.
- While in the Restore state, a Sync message from a given source remains valid for the duration of one $\gamma$. 

\[ t_0 \quad t_0+D \quad t_0+D+d \]

\[ \overline{D} \quad \overline{d} \]
3.2. The Monitor

The messages to be delivered to the destination nodes are deposited on communication channels. A node consists of a state machine and a set of monitors. To assess the behavior of other nodes, a node employs \((K-1)\) monitors, with one monitor for each source of incoming messages, as shown in Figure 2. A node neither uses nor monitors its own messages. The distributed observation of other nodes localizes error detection of incoming messages to their corresponding monitors, and allows for modularization and distribution of the self-stabilization protocol process within a node. A monitor keeps track of the activities of its corresponding source node. Specifically, a monitor reads, evaluates, time-stamps, validates, and stores the last message it receives from that node. A monitor maintains a logical timer, MessageTimer, by incrementing it once per local clock tick. This timer is reset upon receiving a Sync message. A monitor also disposes retained valid messages as indicated by the protocol (Sections 4).

![Figure 2. The i\(^{th}\) node, \(N_i\), with its monitors and state machine.](image)

3.3. The State Machine

The assessment results of the monitored nodes are utilized by the node in the self-stabilization process. The state machine has two states, Restore \((R)\) and Maintain \((M)\), that reflect the current state of the node in the system as shown in Figure 3. The state machine triggers a Sync message broadcast when it transitions from the Restore state to the Maintain state. The state machine describes the behavior of the node, \(N_i\), utilizing assessment results from its monitors, \(M_1 \ldots M_{i-1} M_{i+1} \ldots M_k\) as shown in Figure 2, where \(M_j\) is the monitor for the corresponding node \(N_j\). In addition to the behavior of its corresponding source node, a monitor’s internal status is influenced by the current state of the node’s state machine. When the state machine transitions to the Restore state, the monitors update their internal status as appropriate (Section 3.4).
The **transitory conditions** enable the node to migrate from the *Restore* state to the *Maintain* state. Although during the self-stabilization process a node may transition from the *Restore* state to the *Maintain* state upon a timeout, during steady state such a timeout is indicative of an abnormal situation. The **transitory conditions** are defined with respect to the steady state where such timeouts do not occur. The **transitory delay** is the length of time a node stays in the *Restore* state. The minimum required duration for the transitory delay is denoted by $TD_{min}$, and the maximum duration by $TD_{max}$. $TD_{min}$ is a derived parameter and a function of $F$. For the fully connected topology considered here, the transitory conditions are defined as follows.

1. The node has remained in the *Restore* state for at least $TD_{min}$, where
   
   $TD_{min} = 2$, for $F = 0$, or
   
   $TD_{min} = 2F$, for $F > 0$, and

2. One $\gamma$ has passed since the arrival of the last valid *Sync* message.

The maximum duration of the transitory delay, denoted $TD_{max}$, after meeting the $TD_{min}$ requirement depends on the number of additional valid *Sync* messages received and the drift rate $\rho$. The upper bound for $TD_{max}$ during steady state will be determined later in this report.

In the *Restore* state, the node will either meet the transitory conditions and transition to the *Maintain* state, or remain in the *Restore* state for a predetermined maximum duration until it times out and then transitions to the *Maintain* state. In the *Maintain* state, a node will either transition to the *Restore* state when at least $T_R$ other nodes have transitioned out of the *Maintain* state as indicated by the reception of at least $T_R$ valid *Sync* messages, or remain in the *Maintain* state for a predetermined maximum duration until it times out and transitions to the *Restore* state. The derived parameter $T_R$ is defined as $T_R = F + 1$ and is used as a threshold in conjunction with the *Sync* messages.

In Figure 4 the transitions of a good node to the *Restore* state and from the *Restore* state to the *Maintain* state (during steady state) are depicted along a timeline of node activities. A *Sync* message is transmitted as the node transitions from the *Maintain* state to the *Restore* state. Activities of the *StateMachine* and *LocalTimer* of the node as it transitions between different states are also depicted in this figure.
The clocks need to be periodically synchronized due to their inherent drift with respect to each other. The periodic synchronization during steady state is referred to as the resynchronization process, whereby all good nodes transition to the Restore state and then synchronously to the Maintain state. The resynchronization process begins when the first good node transitions to the Restore state and ends after the last good node transitions to the Maintain state. An upper bound on the duration of the resynchronization process will be determined later in this report.

The synchronization period during steady state, denoted $P$, is defined as the largest time interval between two consecutive resets of the LocalTimer by a good node. The synchronization period depends on the maximum duration of both states of the state machine. The maximum duration for the Restore state is denoted by $P_R$, and the maximum duration for the Maintain state is denoted by $P_M$, where $P_R$ and $P_M$ are expressed in terms of $\gamma$. The length of time that a good node stays in the Restore state is denoted by $L_R$. During steady state $L_R$ is always less than $P_R$. The length of time a good node stays in the Maintain state is denoted by $L_M$. The synchronization period, $P$, is defined by $P = P_R + P_M$ and is expressed in terms of $\gamma$. The actual synchronization period, $P_{Actual}$, is the time interval (during steady state) between the last two consecutive resets of the LocalTimer of a good node, where $P_{Actual} = L_R + L_M < P$.

A node keeps track of time by incrementing its logical time clock StateTimer once every $\gamma$. After the StateTimer reaches $P_R$ or $P_M$, depending on the current state of the node, the node times out, resets the StateTimer, and transitions to the other state. If the node was in the Maintain state, it transmits a new Sync message. The current value of this timer reflects the duration of the current state of the node.
This protocol is expected to be used as the fundamental mechanism to bring and maintain a system within a known synchronization precision bound. Therefore, the protocol has to properly filter out inherent oscillations in the StateTimer during the resynchronization process as depicted in Figure 4. This is resolved by using the LocalTimer in the protocol. The LocalTimer is intended to be used by higher level protocols and must be managed properly to provide the desired monotonically increasing value between adjustments. The logical time clock LocalTimer is incremented once every local clock tick and is reset either when it reaches its maximum allowed value or when the node has transitioned to the Maintain state and remained in that state for ResetLocalTimerAt local clock ticks, where ResetLocalTimerAt is constrained by the following inequality:

\[
\lceil \frac{\Delta_{\text{Precision}}}{\gamma} \rceil \leq \text{ResetLocalTimerAt} \leq P_M - \lceil \frac{\Delta_{\text{Precision}}}{\gamma} \rceil
\]  

(1)

The synchronization precision, denoted \(\Delta_{\text{Precision}}\), is the guaranteed upper bound on the maximum separation between the LocalTimers of any two good nodes. The ResetLocalTimerAt can be given any value in the range specified in inequality (1). However, the value must be the same at all good nodes. In this equality, the lower bound indicates when all good nodes have transitioned to the Maintain state and the upper bound indicates when the first node might transition out of the Maintain state. We choose the earliest such value, \(\text{ResetLocalTimerAt} = \lceil \frac{\Delta_{\text{Precision}}}{\gamma} \rceil\), to reset the LocalTimer of all good nodes. Any value greater than \(\lceil \frac{\Delta_{\text{Precision}}}{\gamma} \rceil\) will prolong the convergence time.

The LocalTimer is also used in assessing the state of the system in the resynchronization process and is bounded by \(P \cdot \gamma\). During steady state, the value of LocalTimer is always less than \(P \cdot \gamma\).

3.4. Protocol Functions

The functions used in the protocol are described in this section.

The function InvalidSync() is used by the monitors. This function determines whether a received Sync message is invalid. When this function returns a true value, it indicates that an unexpected behavior by the corresponding source node has been detected.

The function ConsumeMessage() is used by the monitors. When the host node is in the Restore state, the monitor invalidates the stored Sync message after it has been kept for one \(\gamma\).

The Retry() function determines if at least \(T_R\) other nodes have transitioned out of the Maintain state, where \(T_R = F + 1\). When at least \(T_R\) valid Sync messages from as many nodes have been received, this function returns a true value indicating that at least one good node has transitioned to the Restore state. This function is used to transition from the Maintain state to the Restore state.

The TransitoryConditionsMet() function determines proper timing of the transition from the Restore state to the Maintain state. This function keeps track of the passage of time by
monitoring the StateTimer and determines if the node has been in the Restore state for at least \( T D_{\text{min}} \). It returns a true value when the transitory conditions are met.

The TimeOutRestore() function asserts a timeout condition when the value of the StateTimer has reached \( P_R \) in the Restore state. Such timeout triggers the node to transition to the Maintain state.

The TimeOutMaintain() function asserts a timeout condition when the value of the StateTimer has reached \( P_M \) in the Maintain state. Such timeout triggers the node to reengage in another round of the resynchronization process.

In addition to the above functions, the state machine utilizes the TimeOutGammaTimer() function, which is used to regulate node activities at the \( \gamma \) boundaries. It maintains a GammaTimer by incrementing it once per local clock tick. Once the value of the GammaTimer reaches \( \gamma \) it is reset and the function returns a true value.

### 3.5. Protocol Assumptions

The protocol assumptions are as follows.

1. The cause of transient faults has dissipated.
2. At most \( F \) of the nodes remain faulty.
3. All good nodes correctly execute the protocol.
4. The source of a message is uniquely identifiable by the receivers.
5. A message sent by a good node will be received and processed by all other good nodes within \( \gamma \), where \( \gamma \geq (D + d) \).
6. The initial values of the variables of a node can be set to arbitrary values within their corresponding range. (In an implementation, it is expected that some local mechanism exists to enforce type consistency for all variables.)

### 3.6. The Self-Stabilizing Clock Synchronization Problem

To simplify the presentation of this protocol, it is assumed that all time references are with respect to an initial real time \( t_0 \), where \( t_0 = 0 \) when the protocol assumptions are satisfied, and for all \( t > t_0 \) the system operates within the protocol assumptions.

We define the following symbols:

- \( C \) denotes a bound on the maximum convergence time,
- \( \Delta_{\text{LocalTimer}}(t) \), for real time \( t \), is the maximum difference of values of the LocalTimers of any two good nodes, and
- \( \Delta_{\text{Precision}} \), the synchronization precision, is the guaranteed upper bound on \( \Delta_{\text{LocalTimer}}(t) \).

The maximum difference in the value of LocalTimer for all pairs of good nodes at time \( t \), \( \Delta_{\text{LocalTimer}}(t) \), is determined by the following equation while accounting for the variations in the values of the LocalTimer, across all good nodes.

\[
\Delta_{\text{LocalTimer}}(t) = \min \left( (\text{LocalTimer}_{\text{max}}(t) - \text{LocalTimer}_{\text{min}}(t)), (\text{LocalTimer}_{\text{max}}(t - r) - \text{LocalTimer}_{\text{min}}(t - r)) \right),
\]
where,
\[ r = \lceil \Delta_{\text{Precision}} / \gamma \rceil \]
\[ \text{LocalTimer}_{\text{min}}(x) = \min (\text{LocalTimer}_i(x)), \]
\[ \text{LocalTimer}_{\text{max}}(x) = \max (\text{LocalTimer}_i(x)). \]

There exist \( C \) and \( \Delta_{\text{Precision}} \) such that the following self-stabilization properties hold.
1. **Convergence:** \[ \Delta_{\text{LocalTimer}}(C) \leq \Delta_{\text{Precision}} \]
2. **Closure:** \[ \forall t \geq C, \; \Delta_{\text{LocalTimer}}(t) \leq \Delta_{\text{Precision}} \]
3. **Congruence:** \[ \forall \text{ good nodes } N_i \text{ and } N_j, \; \forall t \geq C, \; \text{LocalTimer}_i(t) = 0 \Rightarrow N_i \text{ and } N_j \text{ are in the Maintain state.} \]

The values of \( \Delta_{\text{SS, min}}, C, \Delta_{\text{Precision}}, \) and the maximum value for \( \text{LocalTimer}, P, \) are determined to be:
\[ \Delta_{\text{SS, min}} = (TD_{\text{min}} \cdot \gamma + 1), \]
\[ C = (2P_R + P_M) \cdot \gamma, \]
\[ \Delta_{\text{Precision}} = (3F - 1) \cdot \gamma - D + \Delta_{\text{Drift}}, \]
\[ P = P_R + P_M, \]
\[ P_M \gg P_R, \]

where the amount of drift from the initial precision is given by
\[ \Delta_{\text{Drift}} = ((1+\rho) - 1/(1+\rho)) \cdot P \cdot \gamma. \]

Note that since \( P > \frac{1}{2} P_R \) and the \( \text{LocalTimer} \) is reset after reaching \( P \) (worst case wraparound), a trivial solution is not possible.
4. The Clock Synchronization Protocol

The protocol presented in Figure 5 consists of a state machine and a set of monitors which execute once every local oscillator tick.

Monitor:

```plaintext
case (incoming message from the corresponding node)
  Resync:
    if InvalidSync() then
      Invalidate the message
    else
      Validate and store the message.
  Other:
    Do nothing.
} // case

ConsumeMessage()
```

Node:

```plaintext
case (state of the node)
  Restore:
    if TimeOutRestore() then
      Reset StateTimer,
      Go to Maintain state,
    else
      if TransitoryConditionsMet() then
        Reset StateTimer,
        Go to Maintain state,
      else
        Stay in Restore state.
  Maintain:
    if TimeOutMaintain() or Retry() then
      Transmit Sync message,
      Reset StateTimer,
      Go to Restore state,
    elseif TimeOutGammaTimer() then
      if (StateTimer = ⌈Δprecision/γ⌉)
        Reset LocalTimer,
        Stay in Maintain state,
      else
        Stay in Maintain state.
} // case
```

† In [Mal 2006A, 2006B, 2007, 2008], upon `TimeOutRestore()`, the node transmitted a Sync message and remained in the Restore state. The modification introduced here simplifies the proof argument and does not change the properties of the protocol.
Semantics of the pseudo-code

- Indentation is used to show a block of sequential statements.
- ‘,’ is used to separate sequential statements.
- ‘.’ is used to end a statement.
- ‘..’ is used to mark the end of a statement and at the same time to separate it from other sequential statements.

5. Proof

The lemmas and theorems are presented in this section. The proof approach is to show that a system of $K \geq 3F + 1$ nodes asynchronously converges from any (un-stabilized) condition to a condition where all good nodes are in the Restore state and then synchronously transition to the Maintain state within a guaranteed initial precision (stabilized). The system is then shown to remain within the timing bounds of the synchronization precision $\Delta_{\text{precision}}$. This idea is depicted in Figure 6.

![Diagram of states](image)

**Figure 6. The proof idea.**

To achieve this goal, first, a good node is shown to transition from the Restore state to the Maintain state and visa versa infinitely often. Second, the analysis consists of three possible scenarios where none, some, or all good nodes are in the Maintain state. Third, the good nodes are shown to transition to the Restore state after the elapse of some time and then synchronously to the Maintain state. Finally, the system is shown to transition between these two states infinitely often while preserving the synchronization precision.

Since the oscillator drift rate, $\rho$, does not play a significant role in the convergence process, it is omitted from the expressions regarding parameters, constants, equations, and the proofs of convergence. However, $\rho$ does affect the closure property and is included in expressions regarding $\Delta_{\text{precision}}$. Omission of $\rho$ does not change the behavior of the protocol or the validity of the proofs. The effect of $\rho$ is later visited to show that its omission in the convergence process as well as the resynchronization process is justified.

Throughout the proofs, the protocol assumptions apply and unless stated otherwise, all references to the Sync messages are with respect to valid Sync messages.

A node behaves **properly** if it correctly executes the protocol.
Lemma MaintainWithinPR – A good node in the Restore state transitions to the Maintain state within at most PR.

Proof – It follows from the protocol that a node in the Restore state will transition to the Maintain state either after meeting the transitory conditions as expressed in function TransitoryConditionsMet(), or because of a resynchronization timeout, as expressed in function TimeOutRestore(). Therefore, a node transitions to the Maintain state within at most PR.

Lemma RestoreWithinPM – A good node in the Maintain state transitions to the Restore state within at most PM.

Proof – It follows from the protocol that a node in the Maintain state will transition to the Restore state either because of a resynchronization timeout, as expressed in function TimeOutMaintain(), or when at least TR other nodes have transitioned out of the Maintain state, as expressed in function Retry(). Since the longest such time interval is bounded by the timeout, the node transitions to the Restore state and transmits a Sync message in at most PM.

Lemma ShortestRestore – The minimum duration of the Restore state is TDmin·γ.

Proof – From the definition of the transitory conditions, a node has to remain in the Restore state for at least TDmin·γ. It also follows that if no valid Sync messages arrive during last γ the node will transition to the Maintain state at the end of this time interval. Hence, the minimum duration of the Restore state is TDmin·γ.

Lemma DeltaSSmin – The minimum time interval between any two consecutive Sync messages from a good node is ΔSS,min = TDmin·γ + 1 clock ticks.

Proof – A node transmits a Sync message when it enters the Restore state. The amount of time the node stays in the Maintain state is defined as ΔMR and depicted in Figure 7. From Lemma ShortestRestore, the minimum duration of the Restore state is TDmin·γ. The time separation between any two consecutive Sync messages from a good node is given by ΔSS ≥ TDmin·γ + ΔMR clock ticks. Since the message processing time is non-zero, ΔMR ≥ 1 clock tick, and therefore, ΔSS,min = TDmin·γ + 1 clock ticks.

![Figure 7. Shortest Maintain state.](image)

All good nodes validate a Sync message from a good node if the time interval between consecutive messages, i.e., ΔSS,min, is not violated. By Lemma DeltaSSmin, consecutive Sync messages from a good node are always more than TDmin apart. Therefore, a message transmitted by a good node after ΔSS,min clock ticks from a random start is guaranteed to be valid. If a node is in the Restore state, from lemma MaintainWithinPR, it will transition to the Maintain state within
$P_R$. For now, let $P_R > 6F$. We will determine the minimum value for $P_R$ later in this report. Since $P_R$ is larger than $\Delta_{SS,min}$, after $P_R$ from a random start, all Sync messages from a good node are at least $\Delta_{SS,min}$ apart and meet the timing requirements at the receiving good nodes. Therefore, the **pre-convergence conditions** are defined as:

1. Time has elapsed for at least $P_R$ from a random start, i.e., $t \geq t_0 + P_R$.
2. All Sync messages from the good nodes are valid at the receiving good nodes (Lemma $\text{DeltaSSmin}$).

Thus, for the following lemmas and theorems, the state of the system is considered after the **pre-convergence conditions** are met. At this point, the system is in one of the following three states:

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

The approach for the proof is depicted in Figure 8. The system is shown to converge from any state and upon convergence maintain the closure property. The figure is partitioned via two dashed lines into three regions. The left region depicts the *pre-convergence conditions* and in conjunction with the middle region they depict the state of the system in the convergence process. The right region depicts the system operating in *steady state* and maintaining the synchronization precision. In this figure, the states **All**, **Some**, and **None** represent the three possible cases from a random start when the **pre-convergence conditions** are met. The propositions associated with each edge indicate that a transition from one state to another may eventually take place.

![Figure 8. Approach for proof.](image-url)
**Theorem** ConvergeNoneMaintain – A system of $K \geq 3F + 1$ nodes satisfying the pre-convergence conditions, where none of the good nodes are in the Maintain state and for all good nodes, $\text{StateTimer}(t) < P_R - 3F - 1$, will always converge to within an initial precision.

Proof – Since none of the good nodes are in the Maintain state, they are in the Restore state because they have not met the transitory conditions. We consider the system at the point of transmission of a Sync message by the last good node in the Maintain state. After the last good node transitions to the Restore state, transitioning of the good nodes back to the Maintain state can further be delayed only upon receiving valid Sync messages from the faulty nodes. Since there are up to $F$ faulty nodes in the system and if their valid Sync messages are $\gamma$ apart from each other (worst case due to transitory condition 2), all good nodes will transition to the Maintain state within at most $TD_{\text{min}} + F = 3F$ (assuming $F > 0$) of the last good node’s Sync message. Since all good nodes are in the Restore state, after receiving $F$ valid Sync messages from as many faulty nodes, any subsequent Sync messages from the faulty nodes will arrive within $TD_{\text{min}}$ and, thus, will be deemed invalid. As a result, for all good nodes, $\text{StateTimer}(t) < P_R - 1$ and they will transition to the Maintain state within the next $\gamma$ without timing out.

![Figure 9. EM and LM for all $F \geq 0$.](image)

The earliest a good node transitions to the Maintain state (EM) is at $t_{\text{EM}}$ after it has remained in the Restore state for the minimum duration of the transitory delay (transitory condition 1) and one $\gamma$ after receiving the last Sync message from the last good node that transitioned to the Restore state (transitory condition 2). Since $\Delta_{\text{SS}} > TD_{\text{min}} \cdot \gamma$, consecutive valid Sync messages from a faulty node are more than $TD_{\text{min}}$ apart. Therefore, locally there will always be a gap of one $\gamma$ interval without a valid Sync message. As a result, a good node will meet the transitory conditions and transition to the Maintain state.

As depicted in Figure 9, since the earliest the last Sync message can arrive at $N_{\text{EM}}$ node is $D$ ticks after the transmission of the last valid Sync message, therefore, $EM = D + \gamma$. The latest a good node transitions to the Maintain state (LM) is at $t_{\text{LM}}$ after remaining in the Restore state for $TD_{\text{min}} + F$, i.e., after receiving valid Sync messages from all faulty nodes. In this case, the $t_{\text{EM}}$ happens at the time the last good node transmits the Sync message, i.e., at $t_{\text{LM}}$ since its transition to the Restore state. So, $LM = (TD_{\text{min}} + F) \cdot \gamma = 3F \cdot \gamma$. Thus, the time difference between the $N_{\text{LM}}$ and $N_{\text{EM}}$ is given by:

\[
\text{StateTimer}_{\text{LM}}(t) - \text{StateTimer}_{\text{EM}}(t) = \Delta_{\text{LMEM}}.
\]

\[
\Delta_{\text{LMEM}} = LM - EM = 3F \cdot \gamma - (D + \gamma) = (3F - 1) \cdot \gamma - D.
\]
Therefore, such a system always converges to within the initial precision of $\Delta_{\text{LMEM}}$.

The synchronization precision $\Delta_{\text{Precision}}$ is the maximum time difference between the LocalTimer of any two good nodes when the system is synchronized. It is the guaranteed precision of the protocol. From theorem ConvergeNoneMaintain, the initial guaranteed precision after the resynchronization is the maximum value of $\Delta_{\text{LMEM}}$. After the initial synchrony the LocalTimers of the good nodes will deviate from the initial precision due to the drift rate of the oscillators. This phenomenon is depicted in Figure 10.

$$\Delta_{\text{Precision}} = \Delta_{\text{LMEM}} + \Delta_{\text{Drift}},$$

where, the amount of drift from the initial precision is given by

$$\Delta_{\text{Drift}} = ((1+\rho) - 1/(1+\rho)) \cdot P \cdot \gamma,$$

where, $P = P_R + P_M$. The factors $(1+\rho)$ and $1/(1+\rho)$ are bounds for the drift of the slowest and fastest nodes in the system, respectively. Therefore,

$$\Delta_{\text{Precision}} = (3F - 1) \cdot \gamma - D + \Delta_{\text{Drift}}.$$ 

**Theorem** ConvergeAllMaintain – A system of $K \geq 3F + 1$ nodes satisfying the pre-convergence conditions, where all good nodes are in the Maintain state, will always converge.

Proof – Since no assumptions are made about the initial relative timing of the good nodes, $\Delta_{\text{LocalTimer}}(t) > \Delta_{\text{Precision}}$ is possible. It follows from the protocol and lemma RestoreWithin$P_M$ that a good node will transition to the Restore state and transmit a Sync message within $P_M$. A good node in the Maintain state keeps track of other nodes that have transitioned to the Restore state. We consider the system after $(T_R - 1)$ good nodes have transitioned to the Restore state. There are two possible scenarios. In the first scenario, the first $(T_R - 1)$ good nodes that transitioned to the Restore state remain in the Restore state until the $T_R$th good node transitions to the Restore state. In the second scenario, some (or all) of the first $(T_R - 1)$ good nodes that had transitioned to the Restore state remain in the Restore state while others (or all) transition back to the Maintain state before the $T_R$th good node transitions to the Restore state. In either case, after the $T_R$th good node transitions to the Restore state, the remaining good nodes in the Maintain state, at least $F$, receive $T_R$ valid Sync messages from as many good nodes, and transmit Sync messages as they transition to the Restore state within the next $\gamma$. 

15
At this point, for the first \((T_R - 1)\) good nodes that remained in the \(\text{Restore}\) state, \(\text{StateTimer}(t) \leq (TD_{min} - 1) + F + T_R = 4F < P_R - F - 1\). The longest duration is \(4F > TD_{min}\), thus, such a node has met the first of the \textit{transitory conditions}. Those good nodes of the first \((T_R - 1)\) good nodes that had transitioned to the \(\text{Restore}\) state and then back to the \(\text{Maintain}\) state will now receive at least \((T_R + 1)\) \(\text{Sync}\) messages within \(2\gamma\) from as many good nodes, will transmit \(\text{Sync}\) messages, and transition to the \(\text{Restore}\) state within the next \(\gamma\). These nodes are within \(2\gamma\) of the recently transitioned good nodes, in particular the \(T_R\)th good node, with \(\text{StateTimer}(t) \leq 2 < P_R - 3F - 1\), and none of them has met the \textit{transitory conditions}. Therefore, the system consists of all good nodes in the \(\text{Restore}\) state with various values for their \(\text{StateTimers}\). At one end of the spectrum, some good nodes have not met the \textit{transitory conditions} with \(\text{StateTimer}(t) < P_R - 3F - 1\). In a similar argument as in theorem \textit{ConvergeNoneMaintain}, since there are up to \(F\) faulty nodes in the system and if the \textit{valid Sync} messages are \(\gamma\) apart from each other (worst case due to \textit{transitory condition 2}), these good nodes will transition to the \(\text{Maintain}\) state within at most \(TD_{min} + F = 3F\) of the last good node’s \(\text{Sync}\) message without timing out. Therefore, \(LM = (TD_{min} + F) \cdot \gamma = 3F \cdot \gamma\).

At the other end of the spectrum, some good nodes have met the first of the \textit{transitory conditions} with \(\text{StateTimer}(t) < P_R - F - 1\). Since there are up to \(F\) faulty nodes in the system and if the \textit{valid Sync} messages are \(\gamma\) apart from each other, these good nodes will transition to the \(\text{Maintain}\) state within the next \(F\) of the last good node’s \(\text{Sync}\) message without timing out.

In a similar argument as in theorem \textit{ConvergeNoneMaintain}, \(EM = D + \gamma\) and the time difference between the \(N_{LM}\) and \(N_{EM}\) is given by

\[
\text{StateTimer}_{LM}(t) - \text{StateTimer}_{EM}(t) = \Delta_{LMEM}.
\]

\[
\Delta_{LMEM} = LM - EM = 3F \cdot \gamma - (D + \gamma) = (3F - 1) \cdot \gamma - D.
\]

Therefore, such a system always converges to within the initial precision of \(\Delta_{LMEM}\). ♦

The theorem \textit{ConvergeSomeMaintain} does not make any assumptions about the initial values of the \(\text{StateTimers}\) of the good nodes. Therefore, its proof encompasses the proof of theorem \textit{ConvergeNoneMaintain}. The proof is more complex and we postpone it until subsequent sections where we first address the special cases of \(F = 1\) and \(F = 0\), and then discuss the general case of \(F > 1\). In the meantime, we continue the proof with lemmas and theorems that apply to the general case of \(F > 0\), \(0 \leq \rho << 1\), and \(d \geq 0\).

**Lemma** \textit{PrecisionLargerThanTDmin} – \textit{For} \(F > 1\), \(\Delta_{\text{Precision}} > TD_{min} \cdot \gamma\)

\[
\text{Proof} – \quad \Delta_{\text{Precision}} > TD_{min} \cdot \gamma \quad (3F - 1) \cdot \gamma - D + \Delta_{\text{Drift}} > 2F \cdot \gamma \quad (F - 1) \cdot \gamma + \Delta_{\text{Drift}} > D.
\]

Since \(\gamma > D\), even if \(\Delta_{\text{Drift}} = 0\), the above inequality reduces to \((F - 1) \cdot \gamma > D\).
Thus, $\Delta_{\text{Precision}} > TD_{\text{min}} \cdot \gamma$.

**Corollary** PrecisionTDminF1 – For $F = 1$, $\Delta_{\text{Precision}} > TD_{\text{min}} \cdot \gamma$ if $\Delta_{\text{Drift}} > D$.

Proof –

\[
\Delta_{\text{Precision}} > TD_{\text{min}} \cdot \gamma
\]
\[
(3F - 1) \cdot \gamma - D + \Delta_{\text{Drift}} > 2F \cdot \gamma
\]
\[
(F - 1) \cdot \gamma + \Delta_{\text{Drift}} > D
\]
\[
\Delta_{\text{Drift}} > D.
\]

It follows from Lemma PrecisionLargerThanTDmin and Corollary PrecisionTDminF1 that depending on the amount of drift $\Delta_{\text{Drift}}$, $\Delta_{\text{Precision}}$ can potentially exceed $TD_{\text{min}}$, i.e., $\Delta_{\text{Precision}} > TD_{\text{min}} \cdot \gamma$ and this in turn can result in a disruption in the normal operation of the system. In particular and as depicted in Figure 10, in a synchronized system of $K \geq 3F + 1$ nodes with an initial precision of $\Delta_{\text{LMEM}}$, after elapse of some time, the nodes can drift apart such that $\Delta_{\text{Precision}} > \Delta_{\text{LMEM}}$. Two such cases are depicted in Figures 11 and 12, where the corresponding activities of the StateTimer and LocalTimer of $N_{\text{Fast}}$ are also depicted during the resynchronization process. In Figure 11, the fast nodes, $N_{\text{Fast}}$, and the slow nodes, $N_{\text{Slow}}$, are less than $\Delta_{\text{Precision}}$ apart, i.e., $\Delta_{\text{LocalTimer}}(t) < \Delta_{\text{Precision}}$. In Figure 12, $N_{\text{Fast}}$ and $N_{\text{Slow}}$ are $\Delta_{\text{Precision}}$ apart from each other, i.e., $\Delta_{\text{LocalTimer}}(t) = \Delta_{\text{Precision}}$.

If $N_{\text{Fast}}$ consists of at least $T_R$ good nodes, then as these nodes transition to the Restore state, the remaining good nodes, $N_{\text{Slow}}$, will follow before timing out as depicted in Figure 11. On the other hand, as depicted in Figure 12, if $N_{\text{Fast}}$ consists of up to $(T_R - 1)$ good nodes, as they transition to the Restore state, the remaining good nodes, $N_{\text{Slow}}$, might not follow. In the meantime, assuming $\Delta_{\text{Precision}} > TD_{\text{min}} \cdot \gamma$ and in the absence of faulty messages, the $N_{\text{Fast}}$ nodes will meet the transitory conditions and transition to the Maintain state. As the $N_{\text{Slow}}$ nodes transition to the Restore state, the $N_{\text{Fast}}$ nodes will follow and once again transition to the Restore state. It follows from theorem ConvergeNoneMaintain that such a system always converges. Since a minority of good nodes temporarily diverge but then converge with the rest of the good nodes, this phenomenon is referred to as **momentary-divergence**.

During steady state and resynchronization process, in the absence of a momentary-divergence, the StateTimer oscillates twice as depicted in Figure 11. However, in the presence of a momentary-divergence, the StateTimer oscillates 4 times as depicted in Figure 12. Proper resetting of the LocalTimer during steady state should guarantee that the LocalTimer remains immune to the StateTimer oscillations.
Figure 11. Activities of $N_{Fast}$ during the resynchronization process, $\Delta_{LocalTimer}(t) < \Delta_{Precision}$.

Figure 12. Activities of $N_{Fast}$ during the resynchronization process, $\Delta_{LocalTimer}(t) = \Delta_{Precision}$. 
Theorem ClosureAllMaintain – A system of \( K \geq 3F + 1 \) nodes, where all good nodes have converged such that \( \Delta_{\text{LocalTimer}}(t) \leq \Delta_{\text{Precision}} \) and all are in the Maintain state, shall remain within the synchronization precision \( \Delta_{\text{Precision}} \).

Proof – It follows from the protocol and lemma \( \text{RestoreWithin}_P \) that a good node will transition to the Restore state within \( P_M \). Since all good nodes are in the Maintain state, as they transmit Sync messages, their transitions to the Restore state are recorded by other good nodes. Since the system is synchronized, the good nodes will transition to the Restore state within \( \Delta_{\text{Precision}} \) of each other. The proof proceeds in the following two parts.

If \( \Delta_{\text{Precision}} < TD_{\text{min}} \gamma \) all good nodes will transition to the Restore state before any of them transitions back to the Maintain state. In this case, for all good nodes, \( \text{StateTimer}(t) < P_R - 3F - 1 \) and none of them has met the transitory conditions. It follows from theorem ConvergeNoneMaintain that such a system always converges to within the initial precision of \( \Delta_{\text{LMEM}} \).

On the other hand, if \( \Delta_{\text{Precision}} \geq TD_{\text{min}} \gamma \) it follows from Lemma PrecisionLargerThanTDMin that some good nodes can potentially transition to the Restore state and then to the Maintain state before all good nodes transition to the Restore state. In other words, the system can experience a momentary-divergence. Similar to the proof of theorem ConvergeAllMaintain, we consider the system after \( (T_R - 1) \) good nodes have transitioned to the Restore state. There are two possible scenarios. In the first scenario, the first \( (T_R - 1) \) good nodes that transitioned to the Restore state remain in the Restore state until the \( T_R \)th good node transitions to the Restore state. In the second scenario, some (or all) of the first \( (T_R - 1) \) good nodes that had transitioned to the Restore state remain in the Restore state while others (or all) transition back to the Maintain state before the \( T_R \)th good node transitions to the Restore state. In either case, after the \( T_R \)th good node transitions to the Restore state, the remaining good nodes in the Maintain state, at least \( F \), receive \( T_R \) valid Sync messages from as many good nodes, and transmit Sync messages as they transition to the Restore state within the next \( \gamma \).

At this point, for the first \( (T_R - 1) \) good nodes that remained in the Restore state, \( \text{StateTimer}(t) \leq (TD_{\text{min}} - 1) + F + T_R = 4F < P_R - F - 1 \). The longest duration is \( 4F > TD_{\text{min}} \), thus, such a node has met the first of the transitory conditions. Those good nodes of the first \( (T_R - 1) \) good nodes that had transitioned to the Restore state and then back to the Maintain state will now receive at least \( (T_R + 1) \) Sync messages within \( 2\gamma \) from as many good nodes, will transmit Sync messages, and transition to the Restore state within the next \( \gamma \). These nodes are within \( 2\gamma \) of the recently transitioned good nodes, in particular the \( T_R \)th good node, with \( \text{StateTimer}(t) \leq 2 < P_R - 3F - 1 \), and none of them has met the transitory conditions. Therefore, the system consists of all good nodes in the Restore state with various values for their StateTimers. At one end of the spectrum, some good nodes have not met the transitory conditions with \( \text{StateTimer}(t) < P_R - 3F - 1 \). In a similar argument as in theorem ConvergeNoneMaintain, since there are up to \( F \) faulty nodes in the system and if the valid Sync messages are \( \gamma \) apart from each other (worst case due to transitory condition 2), these good nodes will transition to the Maintain state within at most \( TD_{\text{min}} + F = 3F \) of the last good node’s Sync message without timing out. Therefore, \( LM = (TD_{\text{min}} + F) \cdot \gamma = 3F \cdot \gamma \).
At the other end of the spectrum, some good nodes have met the first of the transitory conditions with $\text{StateTimer}(t) < P_R - F - 1$. Since there are up to $F$ faulty nodes in the system and if the valid Sync messages are $\gamma$ apart from each other, these good nodes will transition to the Maintain state within the next $F$ of the last good node’s Sync message without timing out.

Once again, in a similar argument as in theorem ConvergeNoneMaintain, $EM = D + \gamma$, and the time difference between the $N_{LM}$ and $N_{EM}$ is given by

\begin{align*}
\Delta_{LMEM} &= LM - EM = 3F \cdot \gamma - (D + \gamma) = (3F - 1) \cdot \gamma - D.
\end{align*}

Therefore, such a system always converges to within the initial precision of $\Delta_{LMEM}$.

**Lemma** StateTimerLessThanPrecision – During the resynchronization process, in steady state, the maximum value of the StateTimer is always less than the synchronization precision $\Delta_{Precision}$.

Proof – From the protocol, the StateTimer is reset when the node transitions to either the Restore state or the Maintain state. It follows from the proof of theorem ClosureAllMaintain that during momentary-divergence some good nodes transition to the Restore state and then back to the Maintain state before others transition to the Restore state. At time $t$ when the last good node has transitioned to the Restore state, the value of the StateTimer of earlier nodes that are in the Maintain state does not exceed $\Delta_{Precision}$. For these good nodes, $\text{StateTimer}(t) \cdot \gamma = \Delta_{Precision} - TD_{min} \cdot \gamma + (D + d)$.

Since $\gamma \geq D + d$,

\begin{align*}
\text{StateTimer}(t) \cdot \gamma &\leq \Delta_{Precision} - TD_{min} \cdot \gamma + \gamma \\
\text{StateTimer}(t) \cdot \gamma &\leq \Delta_{Precision} - (2F - 1) \cdot \gamma
\end{align*}

and for $F > 0$,

\begin{align*}
\text{StateTimer}(t) \cdot \gamma &< \Delta_{Precision}.
\end{align*}

**Theorem** Congruence – For all good nodes $N_i$ and $N_j$ and for $t \geq C$, $\text{LocalTimer}(t) = 0$ implies that $N_i$ and $N_j$ are in the Maintain state.

Proof – From theorem ConvergeNoneMaintain it follows that at the point of convergence when all good nodes have just transitioned to the Maintain state, the initial precision is $\Delta_{LMEM}$. It follows from Lemma StateTimerLessThanPrecision that, during the resynchronization process, in steady state, even when the system experiences a momentary-divergence, $\text{StateTimer}(t) \cdot \gamma < \Delta_{Precision}$ for all good nodes. Therefore, in steady state, StateTimer can reach $\lceil \Delta_{Precision}/\gamma \rceil$ only when the node has transitioned and remained in the Maintain state. Thus, when $\text{StateTimer}_{EM}(t) = \lceil \Delta_{Precision}/\gamma \rceil$, i.e., when $N_{EM}$ resets its LocalTimer, $N_{LM}$, and hence, all good nodes are in the Maintain state.

**Lemma** MaxTransitoryDelay – During steady state, the maximum time a good node stays in the Restore state is given by $TD_{max} = \Delta_{Precision} + (F + 2) \cdot \gamma$. 

20
Proof – Let a good node, $N_1$, be the first to transition to the \textit{Restore} state. Let $N_1$ remain in that state until all other good nodes transition to the \textit{Restore} state. In a synchronized system, the maximum duration of this time is $\Delta_{\text{Precision}}$. After the last good node transitions to the \textit{Restore} state, $N_1$ receives another \textit{valid} \textit{Sync} message and is forced to remain in the \textit{Restore} state at the next $\gamma$. At the next $\gamma$ following the last \textit{valid} \textit{Sync} message from the last faulty node, no \textit{valid} \textit{Sync} messages will be received, there will be a gap of one $\gamma$ interval without a \textit{valid} \textit{Sync} message, and the node will transition to the \textit{Maintain} state. Therefore, during \textit{steady state}, the maximum duration of the transitory delay, $TD_{\text{max}}$, is given by

$$TD_{\text{max}} = \Delta_{\text{Precision}} + \gamma + F \cdot \gamma + \gamma = \Delta_{\text{Precision}} + (F + 2) \cdot \gamma.$$  

This protocol is intended to be used as the fundamental mechanism in bringing and maintaining a system within a known time synchronization precision. In particular, the \textit{LocalTimer} is intended to be used by higher level mechanisms. Therefore, proper management of the \textit{LocalTimer} is one of the guaranteed services provided by the protocol. The logical time clock \textit{LocalTimer} is incremented once every local clock tick and is reset either when it reaches its maximum allowed value or when the node has transitioned to the \textit{Maintain} state and remained in that state for $\text{ResetLocalTimerAt}$ local clock ticks, where $\text{ResetLocalTimerAt}$ is constrained by inequality (1) as described in Section 3.3. Therefore, $P_M$ and $P_R$ have to be sufficiently large to allow time to reset the \textit{LocalTimer} after the node transitions to the \textit{Maintain} state. Specifically, it follows from Figure 12 and Lemma \textit{MaxTransitoryDelay} that $P_R > \lceil (TD_{\text{max}} + \Delta_{\text{Precision}}) / \gamma \rceil$.

$$P_R > \lceil (TD_{\text{max}} + \Delta_{\text{Precision}}) / \gamma \rceil$$
$$P_R > 2 \lceil \Delta_{\text{Precision}} / \gamma \rceil + (F + 2)$$
$$P_R > 2 (3F - 1) + 2 \lceil (\Delta_{\text{Drift}} - D) / \gamma \rceil + (F + 2)$$
$$P_R > 7F + 1 + 2 \lceil (\Delta_{\text{Drift}} - D) / \gamma \rceil.$$  

(2)

If $0 \leq \Delta_{\text{Drift}} < D$, $P_R > 7F - 1$.

If $\Delta_{\text{Drift}} = D$, $P_R > 7F + 1$.

If $2D > \Delta_{\text{Drift}} > D$, $P_R > 7F + 3$.

In general, for all $F > 0$ and $K \geq 3F + 1$, and to prevent early timeout, $P_R$ is constrained by (2). The maximum duration for the \textit{Maintain} state, $P_M$, is typically much larger than $P_R$. Thus, $P_M$ is constrained by $P_M \geq P_R$.

\textbf{Lemma \textit{MaxResyncDuration} – During steady state, the maximum resynchronization duration, MRD, is given by $MRD = 6F \cdot \gamma + \Delta_{\text{Drift}} - D$.}

Proof – It follows from the first part of theorem \textit{ClosureAllMaintain} that the time interval from when the first good node transitions to the \textit{Restore} state until all good nodes transition to the
Maintain state is given by resynchronization duration (RD), \( RD = \Delta_{\text{Precision}} + \) transmission delay of last message + \( LM \), where \( LM = 3F \cdot \gamma \). So,

\[
\Delta_{\text{Precision}} + D + LM \leq RD \leq \Delta_{\text{Precision}} + \gamma + LM \\
((3F - 1) \cdot \gamma - D + \Delta_{\text{Drift}}) + D + 3F \cdot \gamma \leq RD \leq ((3F - 1) \cdot \gamma - D + \Delta_{\text{Drift}}) + \gamma + 3F \cdot \gamma \\
(6F - 1) \cdot \gamma + \Delta_{\text{Drift}} \leq RD \leq 6F \cdot \gamma + \Delta_{\text{Drift}} - D.
\]

Therefore, the maximum value is

\[
MRD = 6F \cdot \gamma + \Delta_{\text{Drift}} - D.
\]

If the duration of the resynchronization process is small relative to the \( P \), the oscillator drift rate, \( \rho \), does not play a significant role in the resynchronization process. It follows from Lemma MaxResyncDuration that \( MRD < P_R \) for all \( F \geq 0 \) and \( \rho \geq 0 \). Since typically \( P_M >> P_R \) and \( MRD << P \), \( \rho \)'s omission from the expressions regarding parameters, constants, and equations in the proof of the resynchronization process is justified.

**Theorem** LocalTimerWithinPrecision – During steady state, in a system of \( K \geq 3F + 1 \) nodes, \( \Delta_{\text{LocalTimer}}(t) \leq \Delta_{\text{Precision}} \).

Proof – It follows from Lemma StateTimerLessThanPrecision that, during the resynchronization process, in steady state, even when the system experiences a momentary-divergence, the StateTimer never reaches \( \left\lceil \Delta_{\text{Precision}} / \gamma \right\rceil \) and thus the LocalTimer will not be reset during this process. On the other hand, it follows from theorem ClosureAllMaintain that once synchronized the good nodes will remain within \( \Delta_{\text{Precision}} \) of each other. Thus, during steady state, \( \Delta_{\text{LocalTimer}}(t) \leq \Delta_{\text{Precision}} \).

**Lemma** SyncWithinP – A good node transmits a Sync message within at most \( (P_R + P_M) \).

Proof – From lemma MaintainWithinP\(_R\), a node in the Restore state will time out within \( P_R \). So, if a node transitions from the Restore state to the Maintain state before it times out, it had remained in the Restore state for at most \( (P_R - 1) \). From lemma RestoreWithinP\(_M\), the node will time out within \( P_M \). It follows from the protocol that a good node transmits a Sync message upon entering the Restore state. Therefore, within at most \( P_R + P_M = P \) a node transmits a Sync message.

The proof proceeds with the following three parts: \( F = 1, F = 0, \) and \( F > 1 \). The cases of \( F = 1 \) and \( F = 0 \) are special cases, and the case of \( F > 1 \) is the general case.

**5.1. Proof For \( F = 1 \)**

The remainder of the proof for the case of \( F = 1 \) is presented in this section. We first present the proof for the ideal case where the logical timers of the good nodes are in-phase with respect to each other and the network imprecision and the oscillator drift are zero, i.e., \( d = 0 \) and \( \Delta_{\text{Drift}} = 0 \). We then expand the proof to a realizable system in subsequent subsections.
5.1.1. In-Phase Case

In this scenario, the local oscillators and logical timers of all good nodes are assumed to be in-phase with each other, and the network imprecision and the drift are zero. Since $\Delta_{\text{Drift}} = 0$, local oscillators and logical timers remain in-phase with each other. This idea is depicted in Figure 13 where $\gamma = 2$.

![Oscillator and Logical Timer Diagram](image)

Figure 13. Ideal case, transitions of logical timers are in-phase.

**Theorem** ConvergeSomeMaintainF1K4 – A system of $F=1$ and $K=3F+1=4$ nodes satisfying the pre-convergence conditions with some of the good nodes in the Maintain state will always converge.

Proof – The good nodes in the Restore state are there because they have not met the transitory conditions. There are three possible scenarios for the system.

Case 1 - All good nodes in the Restore state transition to the Maintain state before the nodes in the Maintain state transition to the Restore state. It follows from theorem ConvergeAllMaintain that such a system always converges to within $\Delta_{\text{LMEM}}$.

Case 2 - All good nodes in the Maintain state transition to the Restore state at least $(3F - 1)\cdot \gamma$ before the nodes in the Restore state transition to the Maintain state. It follows from theorem ConvergeNoneMaintain that such a system always converges.

Case 3 – Some good nodes transition in and out of the Restore state while others transition in and out of the Maintain state. This scenario encompasses those that are not covered in case 2 above. Since there are three good nodes, let’s consider the system where $N_1$ transitions to the Restore state while $N_2$ transitions to the Maintain state. Therefore, $N_2$ and $N_3$ receive a valid Sync message within the next $\gamma$. Now, let’s consider the following two sub-cases.

Case 3.1 – $N_3$ is in the Maintain state. After either $N_2$ or $N_3$ transitions to the Restore state, the other node receives a total of $T_R$ valid Sync messages and transitions to the Restore state. Recall that receiving a valid Sync message prevents a node from transitioning to the Maintain state (transitory condition 2). In the mean time, if $N_1$ had remained in the Restore state due to not meeting the transitory conditions, then for all good nodes, $\text{StateTimer}(t) < P_R - 3F - 1$, and it follows from theorem ConvergeNoneMaintain that such a system always converges. If $N_1$ had transitioned to the Maintain state before the Sync message from $N_2$ and $N_3$ arrive, $N_1$ receives $T_R$ valid Sync messages and returns to the Restore state. Note that the transitory conditions prevent...
$N_1$ from transitioning to the *Maintain* state while $N_2$ and $N_3$ transition to the *Restore* state. Once again, at this point and for all good nodes, $\text{StateTimer}(t) < P^R - 3F - 1$, and it follows from theorem ConvergeNoneMaintain that such a system always converges.

Table 1. Activities of a system of $K = 4$ nodes and $F = 1$.

<table>
<thead>
<tr>
<th>Time</th>
<th>Node 1</th>
<th>Node 2</th>
<th>Node 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_0 + P^R - 2$</td>
<td>$?^{\infty} \rightarrow 0^{\infty}$, Restore</td>
<td>$?^{\infty} \rightarrow 0^{\infty}$, Maintain</td>
<td>$(P^R - 2)^{\infty}$</td>
</tr>
<tr>
<td>$t_0 + P^R - 1$</td>
<td>$1^{\infty}$</td>
<td>$1^{\infty}$</td>
<td>$(P^R - 1)^{\infty}$</td>
</tr>
<tr>
<td>$t_0 + P^R$</td>
<td>$2^{\infty}$</td>
<td>$2^{\infty} \rightarrow 0^{\infty}$, Restore</td>
<td>$P^R^{\infty} \rightarrow 0^{\infty}$, Restore Due to TimeOutRestore()</td>
</tr>
<tr>
<td>$t_0 + P^R + 1$</td>
<td>$3^{\infty}$</td>
<td>$1^{\infty}$</td>
<td>$1^{\infty} \rightarrow 0^{\infty}$, Restore</td>
</tr>
<tr>
<td>$t_0 + P^R + 2$</td>
<td>$4^{\infty}$</td>
<td>$2^{\infty}$</td>
<td>$1^{\infty}$</td>
</tr>
<tr>
<td>$t_0 + P^R + 3$</td>
<td>$5^{\infty}$</td>
<td>$3^{\infty} \rightarrow 0^{\infty}$, Maintain</td>
<td>$2^{\infty}$</td>
</tr>
<tr>
<td>$t_0 + P^R + 4$</td>
<td>$6^{\infty} \rightarrow 0^{\infty}$, Maintain</td>
<td>$1^{\infty}$</td>
<td>$3^{\infty} \rightarrow 0^{\infty}$, Maintain</td>
</tr>
<tr>
<td>$t_0 + P^R + 5$</td>
<td>$1^{\infty}$</td>
<td>$2^{\infty}$</td>
<td>$1^{\infty}$</td>
</tr>
</tbody>
</table>

Case 3.2 – $N_3$ is in the *Restore* state. For this scenario, the system is analyzed at about $P^R$ from $t_0$ on the time axis (see pre-convergence conditions) when $N_3$ is about to time out. The node activities are described with the help of the above table.

Table 1 is an execution trace of a system with parameters $K = 4$, $F = 1$, with no clock drift, $\Delta_{Dif} = 0$. Nodes 1, 2, and 3 are good nodes, and node 4 is the faulty one. The table has four columns, one for time reference and one for each good node. A row depicts activities of all good nodes at the corresponding time. Cell contents for the node columns consist of a number (corresponding to the value of its `StateTimer`) representing the internal status of a node with the stored messages as superscripts, and a description of the possible action by the node. Symbol ‘x’ represents a received *valid Sync* message and symbol ‘–’ represents no *valid Sync* message received from the corresponding node. The position of superscripts, 1 thru 4, corresponds to the source of the message.

At $(t_0 + P^R - 2)$ in the table, $N_1$ transitions to the *Restore* state while $N_2$ transitions to the *Maintain* state and $N_3$ is in the *Restore* state. There are two possible cases regarding $N_3$. If $N_3$ times out within the next $\gamma$ it will transition to the *Maintain* state and receive and retain the *Sync* message from $N_1$. It follows from Case 3.1 above that this system always converges. This is a trivial case and not shown in the above table.

If $N_3$ times out after consuming the *Sync* message from $N_1$, i.e., at $(t_0 + P^R)$ as listed in the above table, in order to keep $N_1$ in the *Restore* state and prevent the system from synchronizing, $N_1$ has to receive a new *valid Sync* message. If $N_1$ receives a *valid Sync* message from another good node, it will have to be from $N_3$ and it follows from Case 3.1 above that this system always converges. So, let $N_1$ receive a *valid Sync* message from the faulty node. At the next $\gamma$ i.e., at $(t_0 + P^R + 1)$, $N_1$ must receive a message from another good node. Let that message come from $N_2$, i.e., $N_2$ transitions to the *Restore* state due to receiving a message from the faulty node at $(t_0 + P^R)$. Therefore, at $(t_0 + P^R + 1)$, $N_3$ and $N_1$ will have received one new *valid Sync* message and $N_1$ will have to stay in the *Restore* state for the next $\gamma$. To keep $N_1$ in the *Restore* state and prevent the
system from synchronizing, \(N_1\) has to receive a new valid \textit{Sync} message every \(\gamma\). To meet the \(\Delta_{SS,\text{min}}\) timing requirement, the next message has to be from a good node, thus at \((t_0+P_R+1)\), \(N_3\) has to transition to the \textit{Restore} state. In this case all good nodes will be in the \textit{Restore} state with \(\text{StateTimer at two extremes; StateTimer}(t) \leq 3 < (P_R - 1)\) and \(N_1\) and \(N_2\) have met the first of \(\Delta_{\text{SS,constraints}}\) and \(\text{StateTimer}(t) = 0 < (P_R - 3F - 1)\) and \(N_3\) has not met the first of the \(\Delta_{\text{S,transitory}}\). In a similar argument as in the proof of theorems \textit{ConvergeNoneMaintain} and \textit{ConvergeAllMaintain} these nodes will not time out and such a system always converges to within the initial precision of \(\Delta_{\text{LMEM}}\). Otherwise, at \((t_0+P_R+1)\), \(N_1\) and \(N_2\) will transition to the \textit{Maintain} state and the system consists of \(T_R\) good nodes in the \textit{Maintain} state. It follows from \textit{Case 3.1} above that this system always converges.

\textbf{Theorem ConvergeSomeMaintainF1KGT4} – A system of \(F=1\) and \(K > 3F+1\) nodes, satisfying the pre-convergence conditions with some good nodes in the \textit{Maintain} state, will always converge.

This case is a generalization of theorem \textit{ConvergeSomeMaintainF1K4} for \(F > K\). In a similar argument, such a system always converges. We do not provide the details of the proof of this theorem here but would like to point out that the system consists of three sets of good nodes, \(S_1, S_2,\) and \(S_3\), where \(K_i = 1 S_i, i = 1, 2, 3\). Since \(K > 3F + 1, G > 2F + 1\) and at least one set, \(S_1\), has \(K_i \geq T_R\). In other words, the presence of the additional good nodes expedites the convergence process.

\textbf{Theorem ConvergeSomeMaintainF1} – A system of \(F=1\) and \(K \geq 3F+1\) nodes, satisfying the pre-convergence conditions with some good nodes in the \textit{Maintain} state will always converge.

\textbf{Proof} – It follows from theorems \textit{ConvergeSomeMaintainF1K4} and \textit{ConvergeSomeMaintainF1KGT4} that such a system always converges.

\textbf{Theorem StabilizeF1} – A system of \(F = 1\) and \(K \geq 3F + 1\) nodes self-stabilizes from any random state after a finite amount of time.

\textbf{Proof} – The proof of this theorem consists of proving the convergence, closure, and congruence properties as defined in section 3.6. The approach for the proof is to show that a system of \(K \geq 3F + 1\) nodes converges from any condition to a state where all good nodes are in the \textit{Restore} state and then synchronously transition to the \textit{Maintain} state within a guaranteed initial precision. The system is then shown to remain within the timing bounds of the synchronization precision of \(\Delta_{\text{Precision}}\). This idea is depicted in Figure 6.

The approach for the proof is depicted in Figure 8. The system is shown to converge from any state and upon convergence maintain the closure property.

\textbf{Convergence Property} – \(\Delta_{\text{LocalTimer}}(C) \leq \Delta_{\text{Precision}}\).

The proof is done in the following three cases.

\textit{Case 1} – None of the good nodes are in the \textit{Maintain} state.
For all good nodes, if \( \text{StateTimer}(t) < P_R - 3F - 1 \), it follows from theorem ConvergeNoneMaintain that such system always converges. Otherwise, it follows from theorem ConvergeSomeMaintainF1 that such system always converges.

Case 2 – All good nodes are in the Maintain state.
It follows from theorem ConvergeAllMaintain that such system always converges.

Case 3 – Some of the good nodes are in the Maintain state.
It follows from theorem ConvergeSomeMaintainF1 that such system always converges.

Closure Property – \( \forall t \geq C, \Delta_{\text{LocalTimer}}(t) \leq \Delta_{\text{Precision}} \).
It follows from theorems ClosureAllMaintain and LocalTimerWithinPrecision that upon convergence, such system always remains stabilized and \( \Delta_{\text{LocalTimer}}(t) \leq \Delta_{\text{Precision}} \) for \( t \geq C \).

Congruence Property – \( \forall \) good nodes \( N_i \) and \( N_j \), \( \forall t \geq C, \text{LocalTimer}_i(t) = 0 \implies N_i \) and \( N_j \) in the Maintain state.
It follows from theorem Congruence that upon convergence, this property is satisfied.

Therefore, such system always self-stabilizes. ♦

Since this protocol self-stabilizes from any state, initialization and/or reintegration are not treated as special cases. Therefore, a reintegrating node will always be admitted to participate in the self-stabilization process as soon as it becomes active.

Theorem ConvergeTime – A system of \( K \geq 3F + 1 \) nodes and \( F \leq 1 \) converges from any random state to a stabilized state within \( C = (2P_R + P_M) \cdot \gamma \).

Proof – In order for the system to stabilize, all good nodes must undergo the resynchronization process. It follows from Lemma SyncWithinP that a good node initiates this process by transmitting a Sync message within at most \( P \). It follows from theorem StabilizeF1 that the system always converges. Also, it follows from that theorem and Lemma MaxResyncDuration that the system converges and all good nodes will transition to the Maintain state at the end of the resynchronization process and within the next \( P_R \). Therefore, the system converges within at most \((P_R + P_M) + P_R\cdot \gamma \) and \( C = (2P_R + P_M)\cdot \gamma \). ♦

Since \( P_{\text{Actual}} < P \) and typically \( P_M \gg P_R \), the maximum convergence time, \( C \), can be approximated to \( C \approx P \). Therefore, \( C \) is a linear function of \( P \) and, similarly, of \( P_M \).

5.1.2. Out-of-Phase Case

The out-of-phase scenario is defined as a system where the logical timers of the good nodes are out of phase with each other, but the local oscillators are in-phase, and the network imprecision and the oscillator drift are also zero, i.e., \( d = 0 \) and \( \Delta_{\text{Drift}} = 0 \). In this scenario, since \( \Delta_{\text{Drift}} = 0 \), the local oscillators of all good nodes remain in-phase with each other. This idea is depicted in the following figure where \( \gamma = 2 \).
Figure 14. Ideal case, transition of logical timers are out-of-phase.

In this scenario, \( d = 0 \) and \( \Delta_{\text{Drift}} = 0 \), therefore, \( \gamma = D \). We do not provide a paper-and-pencil proof of the out-of-phase scenario here. The proof follows the same line of reasoning as above.

A system of \( K = 4 \) nodes for \( \gamma = 1, 2, 3, \) and \( 4 \) were model checked and proven to self-stabilize in the presence of one arbitrary faulty node as expected. Details of the model checking effort for this scenario will be the subject of a subsequent report.

### 5.1.3. A Realizable System

A realizable system is defined as a system where the logical timers and the local oscillators of the good nodes have unconstrained relative phases, and the network imprecision and the oscillator drift are not constrained to be zero, i.e., \( d \geq 0 \) and \( \Delta_{\text{Drift}} \geq 0 \). In this scenario, no assumptions are made about the relative phase difference of local oscillators and logical timers of the good nodes. The local oscillators and logical timers of all good nodes may drift apart with respect to each other. Here, we constrain a realizable system such that \( 0 \leq d \leq 1 \). For such a system we also constrain \( D \geq 1 \). Therefore, \( \gamma \geq 2 \). A paper-and-pencil proof for this scenario is more complex and is left for future work.

Nevertheless, we focus our attention here on the model checking results and report on the issues associated with mapping this protocol to a real system. Since for model checking purposes \( d \) is treated as an integer [Mal 2007, 2008], its value is randomly selected to be either 0 or 1 for a given transmission of a \textit{Sync} message. Several such systems with \( d = \{0, 1\} \), \( D = 1, 2, \) or \( 3 \), and \( \gamma = 2, 3, \) or \( 4 \), respectively, were model checked and proven to self-stabilize in the presence of one arbitrary faulty node. Details of the model checking of this scenario will be the subject of a subsequent report. The results reveal that for such a realizable system to self-stabilize two additional good nodes are needed. As shown in the following counterexample, a system of \( K = 4, d = \{0, 1\}, D = 1, \) and \( \gamma = 2 \) does not self-stabilize.
Table 2. Counterexample for a system of \( K = 4 \) nodes and \( F = 1 \).

<table>
<thead>
<tr>
<th>Time</th>
<th>Node 1</th>
<th>Node 2</th>
<th>Node 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t + 0 )</td>
<td>( 1^{xx} \rightarrow 0^{xx} ), Restore</td>
<td>( 4^{xx} )</td>
<td>( 1^{xx} )</td>
</tr>
<tr>
<td>( t + 1 )</td>
<td>( 0^{xx} )</td>
<td>( 4^{xx} \rightarrow 0^{xx} ), Maintain</td>
<td>( 1^{xx} )</td>
</tr>
<tr>
<td>( t + 2 )</td>
<td>( 1^{xx} )</td>
<td>( 0^{xx} ), Maintain</td>
<td>( 2^{xx} )</td>
</tr>
<tr>
<td>( t + 3 )</td>
<td>( 1^{xx} )</td>
<td>( 1^{xx} )</td>
<td>( 2^{xx} )</td>
</tr>
<tr>
<td>( t + 4 )</td>
<td>( 2^{xx} \rightarrow 0^{xx} ), Maintain</td>
<td>( 1^{xx} \rightarrow 0^{xx} ), Restore</td>
<td>( 3^{xx} )</td>
</tr>
<tr>
<td>( t + 5 )</td>
<td>( 0^{xx} )</td>
<td>( 0^{xx} )</td>
<td>( 3^{xx} )</td>
</tr>
<tr>
<td>( t + 6 )</td>
<td>( 1^{xx} \rightarrow 0^{xx} ), Restore</td>
<td>( 1^{xx} )</td>
<td>( 4^{xx} )</td>
</tr>
<tr>
<td>( t + 7 )</td>
<td>( 0^{xx} )</td>
<td>( 1^{xx} )</td>
<td>( 4^{xx} \rightarrow 0^{xx} ), Maintain</td>
</tr>
<tr>
<td>( t + 8 )</td>
<td>( 1^{xx} )</td>
<td>( 2^{xx} )</td>
<td>( 0^{xx} )</td>
</tr>
<tr>
<td>( t + 9 )</td>
<td>( 1^{xx} )</td>
<td>( 2^{xx} )</td>
<td>( 1^{xx} )</td>
</tr>
<tr>
<td>( t + 10 )</td>
<td>( 2^{xx} \rightarrow 0^{xx} ), Maintain</td>
<td>( 3^{xx} )</td>
<td>( 1^{xx} \rightarrow 0^{xx} ), Restore</td>
</tr>
<tr>
<td>( t + 11 )</td>
<td>( 0^{xx} )</td>
<td>( 3^{xx} )</td>
<td>( 0^{xx} )</td>
</tr>
<tr>
<td>( t + 12 )</td>
<td>( 1^{xx} \rightarrow 0^{xx} ), Restore</td>
<td>( 4^{xx} )</td>
<td>( 1^{xx} )</td>
</tr>
</tbody>
</table>

There are two solutions for the above system: either the Byzantine-faulty node is restricted to influence the nodes at greater intervals, or additional good nodes are added to the system.

We define three types of Byzantine faulty behaviors here. Recall that \( \Delta_{SS,min} = TD_{min} \cdot \gamma + 1 \) clock ticks and for \( F = 1 \), \( \Delta_{SS,min} = 2\gamma + 1 \).

- A **type-A** Byzantine faulty node transmits *Sync* messages arbitrarily but at intervals greater than or equal to \( \Delta_{SS,min} \), measured separately at each receiving good node. Note that this type is a redefinition of the Byzantine faulty node behaving arbitrarily at every clock tick but is tailored for this protocol.
- A **type-B** Byzantine faulty node transmits *Sync* messages arbitrarily but at intervals greater than or equal to \( 3\gamma + 1 \), measured separately at each receiving good node.
- A **type-C** Byzantine faulty node transmits *Sync* messages arbitrarily but at intervals greater than or equal to \( 4\gamma + 1 \), measured separately at each receiving good node.

Model checking results indicate that the system of \( K = 4 \) nodes described above self-stabilizes in the presence of a **type-C** Byzantine faulty node. Alternatively, after the addition of another good node, a system of \( K = 5 \) nodes self-stabilizes in the presence of a **type-B** Byzantine faulty node. The following table is a counterexample for a system of \( K = 5 \) nodes and in the presence of a **type-A** Byzantine faulty node.
Table 3. Counterexample for a system of $K = 5$ nodes and $F = 1$.

<table>
<thead>
<tr>
<th>Time</th>
<th>Node 1</th>
<th>Node 2</th>
<th>Node 3</th>
<th>Node 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t + 0$</td>
<td>$1 \rightarrow \text{xx} \rightarrow 0 \rightarrow \text{xx}$, Restore</td>
<td>$4 \rightarrow \text{xx} \rightarrow 0 \rightarrow \text{xx}$, Maintain</td>
<td>$3 \rightarrow \text{xx}$</td>
<td>$1 \rightarrow \text{xx}$</td>
</tr>
<tr>
<td>$t + 1$</td>
<td>$0 \rightarrow \text{xx}$</td>
<td>$0 \rightarrow \text{xx}$</td>
<td>$3 \rightarrow \text{xx}$</td>
<td>$2 \rightarrow \text{xx}$</td>
</tr>
<tr>
<td>$t + 2$</td>
<td>$1 \rightarrow \text{xx}$</td>
<td>$1 \rightarrow \text{xx}$</td>
<td>$4 \rightarrow \text{xx}$</td>
<td>$2 \rightarrow \text{xx}$</td>
</tr>
<tr>
<td>$t + 3$</td>
<td>$1 \rightarrow \text{xx}$</td>
<td>$1 \rightarrow \text{xx} \rightarrow 0 \rightarrow \text{xx}$, Restore</td>
<td>$4 \rightarrow \text{xx} \rightarrow 0 \rightarrow \text{xx}$, Maintain</td>
<td>$3 \rightarrow \text{xx}$</td>
</tr>
<tr>
<td>$t + 4$</td>
<td>$2 \rightarrow \text{xx}$</td>
<td>$0 \rightarrow \text{xx}$</td>
<td>$0 \rightarrow \text{xx}$</td>
<td>$3 \rightarrow \text{xx}$</td>
</tr>
<tr>
<td>$t + 5$</td>
<td>$2 \rightarrow \text{xx}$</td>
<td>$1 \rightarrow \text{xx}$</td>
<td>$1 \rightarrow \text{xx}$</td>
<td>$4 \rightarrow \text{xx}$</td>
</tr>
<tr>
<td>$t + 6$</td>
<td>$3 \rightarrow \text{xx}$</td>
<td>$1 \rightarrow \text{xx}$</td>
<td>$1 \rightarrow \text{xx} \rightarrow 0 \rightarrow \text{xx}$, Restore</td>
<td>$4 \rightarrow \text{xx} \rightarrow 0 \rightarrow \text{xx}$, Maintain</td>
</tr>
<tr>
<td>$t + 7$</td>
<td>$3 \rightarrow \text{xx}$</td>
<td>$2 \rightarrow \text{xx}$</td>
<td>$0 \rightarrow \text{xx}$</td>
<td>$0 \rightarrow \text{xx}$</td>
</tr>
<tr>
<td>$t + 8$</td>
<td>$4 \rightarrow \text{xx}$</td>
<td>$2 \rightarrow \text{xx}$</td>
<td>$1 \rightarrow \text{xx}$</td>
<td>$1 \rightarrow \text{xx}$</td>
</tr>
<tr>
<td>$t + 9$</td>
<td>$4 \rightarrow \text{xx} \rightarrow 0 \rightarrow \text{xx}$, Maintain</td>
<td>$3 \rightarrow \text{xx}$</td>
<td>$1 \rightarrow \text{xx}$</td>
<td>$1 \rightarrow \text{xx} \rightarrow 0 \rightarrow \text{xx}$, Restore</td>
</tr>
<tr>
<td>$t + 10$</td>
<td>$0 \rightarrow \text{xx}$</td>
<td>$3 \rightarrow \text{xx}$</td>
<td>$2 \rightarrow \text{xx}$</td>
<td>$0 \rightarrow \text{xx}$</td>
</tr>
<tr>
<td>$t + 11$</td>
<td>$1 \rightarrow \text{xx}$</td>
<td>$4 \rightarrow \text{xx}$</td>
<td>$2 \rightarrow \text{xx}$</td>
<td>$1 \rightarrow \text{xx}$</td>
</tr>
<tr>
<td>$t + 12$</td>
<td>$1 \rightarrow \text{xx} \rightarrow 0 \rightarrow \text{xx}$, Restore</td>
<td>$4 \rightarrow \text{xx} \rightarrow 0 \rightarrow \text{xx}$, Maintain</td>
<td>$3 \rightarrow \text{xx}$</td>
<td>$1 \rightarrow \text{xx}$</td>
</tr>
</tbody>
</table>

After addition of another good node, a system of $K = 6$ nodes self-stabilizes in the presence of a type-A Byzantine faulty node. The following table is a summary of the conditions that a system in this scenario requires to self-stabilize in the presence of a Byzantine-faulty node.

Table 4. A realizable system with $0 \leq d \leq 1$ and $F = 1$.

<table>
<thead>
<tr>
<th>$K$</th>
<th>Byzantine Node $A_{SS,min}$ Intervals</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>$4\gamma + 1$</td>
</tr>
<tr>
<td>5</td>
<td>$3\gamma + 1$</td>
</tr>
<tr>
<td>6</td>
<td>$2\gamma + 1$</td>
</tr>
</tbody>
</table>

5.2. Proof For $F = 0$

The proof for the case of $F = 1$ readily applies to the special case of $F = 0$ and $K \geq 2$. We present the proof of this special case separately.

**Theorem** ConvergeNoneMaintainF0 – A system of $K \geq 2$ nodes satisfying the pre-convergence conditions with none of the good nodes in the Maintain state and for all good nodes, $StateTimer(t) < P_R - 2$, will always converge to within an initial precision.

**Proof** – The proof is similar to the proof of the general case as presented in theorem ConvergeNoneMaintain. Since none of the good nodes are in the Maintain state, they are in the Restore state because they have not met the transitory conditions. We consider the system at the point of transmission of a Sync message by the last good node in the Maintain state, where for all nodes $StateTimer(t) < P_R - 2$. The nodes will receive one last valid Sync message, will remain in the Restore state for another $\gamma$ and since there are no faulty nodes present, all nodes will transition to the Maintain state within the following $\gamma$. Thus, they will not time out while in the Restore state.
Similar to the argument in the proof of theorem `ConvergeNoneMaintain`, the earliest a good node transitions to the `Maintain` state is at \( t_{EM} \) where \( EM = D + \gamma \). The latest a good node transitions to the `Maintain` state is at \( t_{LM} \) and after remaining in the `Restore` state for \( TD_{\text{min}} = 2 \), i.e., \( LM = 2\gamma \). So, the time difference between the \( N_{LM} \) and \( N_{EM} \) is given by
\[
StateTimer_{LM}(t) - StateTimer_{EM}(t) = \Delta_{LMEM}.
\]
\[
\Delta_{LMEM} = LM - EM = 2\gamma - (D + \gamma) = \gamma - D = d.
\]

Therefore, such a system always converges to within the initial precision of \( \Delta_{LMEM} \).

\[\blacklozenge\]

**Theorem** `ConvergeF0` – A system of \( F = 0 \) and \( K \geq 2 \) good nodes will always converge.

Proof – The proof follows in the following three cases.

Case 1 – All nodes are in the `Maintain` state. Since there are no faulty nodes present, \( T_R = 1 \). Therefore, as soon as one of the nodes transitions to the `Restore` state, all others will follow within the next \( \gamma \). All good nodes will transition to the `Restore` state within \( \gamma \) of each other. At this point, for all good nodes, \( StateTimer(t) < 2 \ll P_R - 2 \) and none of them has met the transitory conditions. Since there are no faulty nodes present, the nodes will transition to the `Maintain` state within the next \( \gamma \) and thus will not time out while in the `Restore` state. It follows from theorem `ConvergeNoneMaintainF0` that such a system always converges to within \( \Delta_{LMEM} \).

Case 2 – All nodes are in the `Restore` state. Since in this case no other assumptions are made, some nodes transition to the `Maintain` state due to timeouts while others by meeting the transitory conditions. Nevertheless, all nodes will transition to the `Maintain` state within \( TD_{\text{min}}\gamma \) of each other. As a result the initial precision is \( TD_{\text{min}}\gamma \). It follows from case 1 that this system will be within the initial precision of \( \Delta_{LMEM} \) within the next synchronization round.

Case 3 – Some nodes are in the `Maintain` state while others are in the `Restore` state. If the nodes in the `Restore` state transition to the `Maintain` state before the nodes in the `Maintain` state time out, then the system will consist of all nodes in the `Maintain` state. It follows from case 1 that such system always converges. Conversely, if the nodes in the `Maintain` state transition to the `Restore` state before the nodes in the `Restore` state time out, then the system will consist of all nodes in the `Restore` state. It follows from case 2 that such system always converges. If some nodes transition to the `Maintain` state due to time out, while at least one other node transitions to the `Restore` state, then since \( T_R = 1 \), all nodes that have transitioned to the `Maintain` state will transition back to the `Restore` state within the next \( \gamma \). It follows from case 2 that such system always converges. Therefore, a system of \( F = 0 \) and \( K \geq 2 \) good nodes will always converge to within \( \Delta_{LMEM} \).

From theorem `ConvergeNoneMaintainF0` the initial guaranteed precision after the resynchronization is the maximum value of \( \Delta_{LMEM} \). For this case, since \( F = 0 \), \( TD_{\text{min}} = 2 \) and
\[
StateTimer_{LM}(t) - StateTimer_{EM}(t) = \Delta_{LMEM}.
\]
\[
\Delta_{LMEM} = LM - EM = (TD_{\text{min}} + F)\gamma - (D + \gamma) = 2\gamma - (D + \gamma) = \gamma - D = d.
\]
Therefore, the system converges to within at most \((\gamma - D) = d\) of each other.

The guaranteed synchronization precision \(\Delta_{\text{Precision}}\) after elapsed time of \(P\) is bounded by,
\[
\Delta_{\text{Precision}} = \Delta_{\text{LMEM}} + \Delta_{\text{Drift}}.
\]

For \(F = 0\),
\[
\Delta_{\text{Precision}} = \Delta_{\text{LMEM}} + \Delta_{\text{Drift}} = d + \Delta_{\text{Drift}}.
\]

**Corollary PrecisionTDminF0** – *For \(F = 0\), \(\Delta_{\text{Precision}} > TD_{\text{min}} \cdot \gamma\) if \(\Delta_{\text{Drift}} > D + \gamma\)*

**Proof** –
\[
\Delta_{\text{Precision}} > TD_{\text{min}} \cdot \gamma \\
(\gamma - D) + \Delta_{\text{Drift}} > 2\gamma \\
\Delta_{\text{Drift}} > D + \gamma
\]

**Theorem ClosureAllMaintainF0** – *A system of \(F = 0\) and \(K \geq 2\) good nodes, where all nodes have converged such that all nodes are in the Maintain state and \(\Delta_{\text{LocalTimer}}(t) \leq \Delta_{\text{Precision}}\), shall remain within the synchronization precision \(\Delta_{\text{Precision}}\).*

**Proof** – Since all good nodes are in the Maintain state, it follows from lemma \(\text{RestoreWithin}P_M\) that upon timeout, the nodes will transmit \(\text{Sync}\) messages and transition to the Restore state within \(P_M\). As they transmit \(\text{Sync}\) messages, their transitions to the Restore state are recorded by other good nodes that are in the Maintain state. Furthermore, since \(T_R = 1\), as soon as one node transitions to the Restore state, the other nodes will transition to the Restore state within the next \(\gamma\). At this point, for all good nodes, \(\text{StateTimer}(t) < 2 \ll P_R - 2\) and none of them has met the transitory conditions. Since there are no faulty nodes present, the nodes will transition to the Maintain state within the next \(\gamma\) and, thus, will not time out while in the Restore state. It follows from theorem \(\text{ConvergeNoneMaintainF0}\) that such a system always converges to within \(\Delta_{\text{LMEM}}\).

**Theorem StabilizeF0** – *A system of \(F = 0\) and \(K \geq 2\) nodes self-stabilizes from any random state after a finite amount of time.*

**Proof** – It follows from theorem \(\text{ConvergeF0}\) that the nodes converge and upon convergence transition to the Maintain state within \(\Delta_{\text{LMEM}}\) of each other. It follows from theorem \(\text{ClosureAllMaintainF0}\) that such system of \(F = 0\) and \(K \geq 2\) nodes always remains within the \(\Delta_{\text{Precision}}\) bounds. Thus, \(\Delta_{\text{LocalTimer}}(t) \leq \Delta_{\text{Precision}}\). It follows from theorem \(\text{Congruence}\) that upon convergence, this property is satisfied. Therefore, such system always self-stabilizes.

**5.3. Generalization Of The Protocol, For \(F > 1\)**

It follows from theorems \(\text{StabilizeF0, ConvergeNoneMaintain, ConvergeAllMaintain,}\) and \(\text{ClosureAllMaintain}\), and their corresponding assumptions that a system under their associated conditions always self-stabilizes for all \(F \geq 0\) and \(0 \leq \rho \ll 1\). It can readily be shown that in the absence of faulty nodes, this protocol always converges from any arbitrary state. Also, if the faults are transient such that the time interval between the consecutive manifestations
of the transient faults is greater than the convergence time \( C \), the system always self-stabilizes for all \( F \geq 0 \). Nevertheless, since theorem \( \text{ConvergeSomeMaintainF1} \) cannot be generalized, this protocol does not seem to solve the general case of clock synchronization for \( F > 1 \). Table 5 is a trace of a counterexample for a system with \( K = 8 \) nodes, where \( F = 2 \) and \( G = 6 \).

Let’s consider the system where some good nodes transition in and out of the Restore state while others transition in and out of the Maintain state. Also, let the system consist of three sets of good nodes, \( S_1, S_2, \) and \( S_3 \), where \( K_i = |S_i|, i = 1, 2, 3 \). For simplicity, let’s assume that all good nodes in a set \( S_i \) are in synchrony with each other such that they all transition from one state to another at the same time. Now let’s consider \( K_1 = K_2 = K_3 = F < T_R \). The following table depicts a scenario that repeats indefinitely and reveals that such a system will not always converge.

Table 5. Activities of a system of \( K \leq 4F \) nodes, \( F > 1 \).

<table>
<thead>
<tr>
<th>Time</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
</tr>
</thead>
<tbody>
<tr>
<td>t + 0</td>
<td>( 2^{x-x} )</td>
<td>( 2^{x-x} )</td>
<td>( 5^{x-x} \rightarrow 0^{x-x}, \text{Maintain} )</td>
</tr>
<tr>
<td>t + 1</td>
<td>( 3^{x-x} )</td>
<td>( 3^{x-x} \rightarrow 0^{x-x}, \text{Restore} )</td>
<td>( 1^{x-x} )</td>
</tr>
<tr>
<td>t + 2</td>
<td>( 4^{x-x} )</td>
<td>( 1^{x-x} )</td>
<td>( 2^{x-x} )</td>
</tr>
<tr>
<td>t + 3</td>
<td>( 5^{x-x} \rightarrow 0^{x-x}, \text{Maintain} )</td>
<td>( 2^{x-x} )</td>
<td>( 3^{x-x} \rightarrow 0^{x-x}, \text{Restore} )</td>
</tr>
<tr>
<td>t + 4</td>
<td>( 1^{x-x} )</td>
<td>( 3^{x-x} )</td>
<td>( 1^{x-x} )</td>
</tr>
<tr>
<td>t + 5</td>
<td>( 2^{x-x} )</td>
<td>( 4^{x-x} )</td>
<td>( 2^{x-x} )</td>
</tr>
<tr>
<td>t + 6</td>
<td>( 3^{x-x} \rightarrow 0^{x-x}, \text{Restore} )</td>
<td>( 5^{x-x} \rightarrow 0^{x-x}, \text{Maintain} )</td>
<td>( 3^{x-x} )</td>
</tr>
<tr>
<td>t + 7</td>
<td>( 1^{x-x} )</td>
<td>( 1^{x-x} )</td>
<td>( 4^{x-x} )</td>
</tr>
<tr>
<td>t + 8</td>
<td>( 2^{x-x} )</td>
<td>( 2^{x-x} )</td>
<td>( 5^{x-x} \rightarrow 0^{x-x}, \text{Maintain} )</td>
</tr>
</tbody>
</table>

The scenario that repeats consists of a set transitioning to the Restore state at the same time another set transitions to the Maintain state while the third set is in the Restore state. For instance, at \( t+6 \), \( S_1 \) transitions to the Restore state, \( S_2 \) transitions to the Maintain state and \( S_3 \) remains in the Restore state. Therefore, during the next \( \gamma \) all sets receive up to \( F \) valid Sync messages. The set \( S_3 \) is forced to remain in the Restore state while \( S_2 \) has up to \( F \) valid Sync messages.

Although this protocol does not solve the general case of this problem, it provides mathematically proven and mechanically verified [Mal 2007, 2008] partial solutions for specific cases of \( F = 0 \) and \( F = 1 \). We intend to use these specific cases as the building blocks for larger and more complex systems.

6. Protocol Overhead

Since only one message, namely Sync, is required for the operation of this protocol, the protocol overhead during steady state is at most (depending on the amount of \( \Delta_{Dij} \)) two messages per \( P \). Also, since only one message is needed, a single binary value is sufficient to represent it.
7. Possible Applications

The proposed self-stabilizing protocol is expected to have many practical applications as well as many theoretical implications. Embedded systems, distributed process control, synchronization, fault tolerance with Byzantine agreement, computer networks, the Internet, Internet applications, security, safety, automotive, aircraft, wired and wireless telecommunications, graph theoretic problems, leader election, time division multiple access (TDMA), and the SPIDER\textsuperscript{1} architecture [Tor 2005A, 2005B] at NASA-LaRC are a few examples. These are some of the many areas of distributed systems that can use self-stabilization in order to design more robust distributed systems.

8. Conclusions

The self-stabilization problem has two facets. It is inherently event-driven and it is also time-driven. Most attempts at solving the self-stabilization problems have focused only on the event-driven aspect of this problem. Additionally, all efforts toward solving this problem must recognize that the system undergoes two distinct phases, un-stabilized and stabilized, and that once stabilized, the system state needs to be preserved. The protocol presented here properly merges the time and event driven aspects of this problem in order to self-stabilize the system in a timely manner. Initialization and/or reintegration are not treated as special cases. These scenarios are regarded as inherent part of this self-stabilizing protocol.

In this report, a rapid Byzantine-fault-tolerant self-stabilizing clock synchronization protocol is presented. The protocol presented here is independent of specific application-dependent requirements and is focused only on clock synchronization of a system in the presence of Byzantine faults and after the cause of transient faults has dissipated. The protocol utilizes a single message, Sync, and during steady state imposes an overhead of at most two messages per synchronization period. A model of this protocol has been mechanically verified using SMV [SMV] where the entire state space has been examined and proven to self-stabilize in the presence of one arbitrary faulty node. Instances of the protocol have been proven to tolerate bursts of transient failures and deterministically converge with a linear time with respect to the synchronization period as predicted. This protocol does not rely on any assumptions about the initial state of the system except for the presence of sufficient number of good nodes, and no assumptions are made about the internal status of the nodes, the monitors, and the communication channels, thus making the weakest assumptions and producing the strongest results. All timing measures of variables are based on the node’s local clock and thus no central clock or externally generated pulse is used. The Byzantine faulty behavior modeled here is a node with arbitrarily malicious behavior. The Byzantine faulty node is allowed to influence other nodes at every clock tick. The only constraint is that the interactions are restricted to defined interfaces.

Proofs of specific instances of this protocol are presented in this report. This protocol has been the subject of a rigorous verification effort. A system tolerating one Byzantine faulty node

\textsuperscript{1} Scalable Processor-Independent Design for Enhanced Reliability (SPIDER).
has been model checked for the in-phase, out-of-phase, and realizable systems. The SMV model checking results verified the correctness of the claims of this self-stabilizing protocol.

Although this protocol does not solve the general case of the problem, it provides proven and verified solutions for specific cases. The paper-and-pencil proofs presented here, in conjunction with the model checking results, indicate that the protocol is applicable to realizable practical systems. We intend to leverage the specific cases as building blocks for larger and more complex systems.

This protocol is intended to be the fundamental mechanism for bringing and maintaining a system within bounded synchrony. Formalization and verification of the integration process of other protocols with this protocol in order to achieve tighter precision are underway. Nevertheless, proper means are embedded in this protocol to accommodate the integration process. Implementation of this protocol in hardware and its characterization in a representative adverse environment are being planned.
References


Appendix A. Symbols

This appendix the symbols used in the protocol.

<table>
<thead>
<tr>
<th>Symbols</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\rho$</td>
<td>bounded drift rate with respect to real time</td>
</tr>
<tr>
<td>$d$</td>
<td>network imprecision</td>
</tr>
<tr>
<td>$D$</td>
<td>event-response delay</td>
</tr>
<tr>
<td>$F$</td>
<td>maximum number of faulty nodes</td>
</tr>
<tr>
<td>$G$</td>
<td>minimum number of good nodes</td>
</tr>
<tr>
<td>$K$</td>
<td>sum of all nodes</td>
</tr>
<tr>
<td>$Sync$</td>
<td>self-stabilization message</td>
</tr>
<tr>
<td>$S$</td>
<td>abbreviation for $Sync$ message</td>
</tr>
<tr>
<td>$\Delta_{SS}$</td>
<td>time difference between the last consecutive $Sync$ messages</td>
</tr>
<tr>
<td>$T_R$</td>
<td>threshold for $Retry()$ function</td>
</tr>
<tr>
<td>$Restore$</td>
<td>self-stabilization state</td>
</tr>
<tr>
<td>$Maintain$</td>
<td>self-stabilization state</td>
</tr>
<tr>
<td>$R$</td>
<td>abbreviation for $Restore$ state</td>
</tr>
<tr>
<td>$M$</td>
<td>abbreviation for $Maintain$ state</td>
</tr>
<tr>
<td>$P_R$</td>
<td>maximum duration while in the $Restore$ state</td>
</tr>
<tr>
<td>$P_{R,\text{min}}$</td>
<td>minimum value of $P_R$</td>
</tr>
<tr>
<td>$P_M$</td>
<td>maximum duration while in the $Maintain$ state</td>
</tr>
<tr>
<td>$P_{\text{Actual}}$</td>
<td>actual synchronization period</td>
</tr>
<tr>
<td>$P$</td>
<td>synchronization period</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>equally spaced time intervals for time-driven activities</td>
</tr>
<tr>
<td>$C$</td>
<td>maximum convergence time</td>
</tr>
<tr>
<td>$\Delta_{\text{LocalTimer}(t)}$</td>
<td>maximum time difference of $LocalTimers$ of any two good nodes at real time $t$</td>
</tr>
<tr>
<td>$LM$</td>
<td>Latest Maintain</td>
</tr>
<tr>
<td>$EM$</td>
<td>Earliest Maintain</td>
</tr>
<tr>
<td>$\Delta_{LMEM}$</td>
<td>difference of $LM$ and $EM$, a.k.a. initial synchronization precision</td>
</tr>
<tr>
<td>$\Delta_{\text{Precision}}$</td>
<td>maximum synchronization precision</td>
</tr>
<tr>
<td>$\Delta_{Drift}$</td>
<td>maximum deviation from the initial synchrony</td>
</tr>
<tr>
<td>$N_i$</td>
<td>the $i^{th}$ node</td>
</tr>
<tr>
<td>$M_i$</td>
<td>the $i^{th}$ monitor of a node</td>
</tr>
</tbody>
</table>
This report presents a rapid Byzantine-fault-tolerant self-stabilizing clock synchronization protocol that is independent of application-specific requirements. It is focused on clock synchronization of a system in the presence of Byzantine faults after the cause of any transient faults has dissipated. A model of this protocol is mechanically verified using the Symbolic Model Verifier (SMV) [SMV] where the entire state space is examined and proven to self-stabilize in the presence of one arbitrary faulty node. Instances of the protocol are proven to tolerate bursts of transient failures and deterministically converge with a linear convergence time with respect to the synchronization period. This protocol does not rely on assumptions about the initial state of the system other than the presence of sufficient number of good nodes. All timing measures of variables are based on the node’s local clock, and no central clock or externally generated pulse is used. The Byzantine faulty behavior modeled here is a node with arbitrarily malicious behavior that is allowed to influence other nodes at every clock tick. The only constraint is that the interactions are restricted to defined interfaces.