NASA/TM-2011-217184

Correctness Proof of a Self-Stabilizing Distributed Clock Synchronization Protocol for Arbitrary Digraphs

Mahyar R. Malekpour
Langley Research Center, Hampton, Virginia

October 2011

NASA/TM-2011-217184

Correctness Proof of a Self-Stabilizing Distributed Clock Synchronization Protocol for Arbitrary Digraphs

Mahyar R. Malekpour
Langley Research Center, Hampton, Virginia

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-2199

October 2011

Acknowledgments

This effort was conducted under the System-Wide Safety and Assurance Technologies (SSAT) Project of NASA's Aviation Safety program. The author would like to thank the reviewers for their in-depth review and helpful comments. The author would like to especially thank George Hagen for his constructive comments.

Available from:

NASA Center for AeroSpace Information
7115 Standard Drive
Hanover, MD 21076-1320
443-757-5802

Abstract

This report presents a deductive proof of a self-stabilizing distributed clock synchronization protocol. It is focused on the distributed clock synchronization of an arbitrary, non-partitioned digraph ranging from fully connected to 1-connected networks of nodes while allowing for differences in the network elements. This protocol does not rely on assumptions about the initial state of the system, and no central clock or a centrally generated signal, pulse, or message is used. Nodes are anonymous, i.e., they do not have unique identities. There is no theoretical limit on the maximum number of participating nodes. The only constraint on the behavior of the node is that the interactions with other nodes are restricted to defined links and interfaces. We present a deductive proof of the correctness of the protocol as it applies to the networks with unidirectional and bidirectional links. We also confirm the claims of determinism and linear convergence.

Table of Contents

1. Introduction

Synchronization algorithms are essential for managing the use of resources and controlling communication in a distributed system. **Synchronization** of a distributed system is the process of **achieving** and **maintaining** a bounded skew among independent local clocks. A distributed system is said to be self-stabilizing if, from an arbitrary state, it is guaranteed to reach a legitimate state in a finite amount of time and remain in a legitimate state. A legitimate state is a state where all parts in the system are in synchrony. The self-stabilizing distributed-system clock synchronization problem is, therefore, to develop an algorithm (i.e., a protocol) to *achieve* and *maintain* synchrony of local clocks in a distributed system after experiencing system-wide disruptions in the presence of network element imperfections. The **convergence** and **closure** properties address achieving and maintaining network synchrony, respectively. Hereafter in this report, we use the term synchronization to mean self-stabilizing clock synchronization in distributed systems.

A thorough understanding of the synchronization of a distributed system has proven to be elusive for decades. The main challenges associated with distributed synchronization are the complexity of developing a solution and proving the correctness of these solutions. It is possible to have a solution that is hard to prove or refute. Such a solution, however, is not likely to be accepted or used in practical systems. The proposed solutions must restore synchrony and coordinated operations after experiencing system-wide disruptions in the presence of network element imperfections and, for ultra-reliable distributed system, in the presence of various faults. A fault is a defect or flaw in a system component resulting in an incorrect state [Gir 2005] [Tor 2005] [But 2008]. In addition, a proposed solution must be proven to be correct. If a mathematical proof is deemed difficult, at a minimum, the proposed solution must be shown to be correct using available formal methods techniques. Furthermore, addressing network element imperfections is necessary to make a solution applicable to realizable systems.

In [Mal 2011A] a solution is presented for an arbitrary network (digraph) in the absence of faults. The system under study is an arbitrary, non-partitioned digraph ranging from fully connected to 1-connected networks of nodes while allowing for differences in the network elements. Some networks of interest include grid, ring, fully connected, bipartite, and star (hub). This solution does not require any particular information flow nor imposes changes (e.g., embedding a directed spanning tree or rewiring) to the network in order to achieve synchrony. The assumption of an absence of faults is equivalent to the assumption that all faults are detectable. This departure from our previous work at the Byzantine extreme of the fault spectrum [Mal 2006A] is in part because of the niche use and the extra cost associated with the Byzantine faults. Also, using authentication and error detection techniques, it is possible to substantially reduce the effects of variety of faults in the system. Furthermore, the classical definition of a self-stabilizing algorithm assumes generally that there are no faults in the system.

In this report we present a deductive proof for the correctness claims of *A Self-Stabilizing Distributed Clock Synchronization Protocol For Arbitrary Digraphs* [Mal 2011A] and claims of determinism and linear convergence of the protocol with respect to the self-stabilization period.

A bounded model of the protocol was model checked by confirming that a set of candidate systems self-stabilized from any state [Mal 2011B]. The model checking results of the bounded models of the protocol have validated the correctness of the protocol as they apply to the networks with unidirectional and bidirectional links. In addition, the results have confirmed the claims of determinism and linear convergence.

This report is organized as follows. In Section 2 we provide a system overview. We present the protocol and its description in Section 3. We present a deductive proof of the correctness of the protocol in Section 4. We present concluding remarks in Section 5.

2. System Overview

We consider a system of pulse-coupled entities (e.g., oscillators, pacemaker cells) pulsating periodically at regular time intervals. These entities are said to be coupled through some physical means (wire or fiber cables, chemical process, or wirelessly through air or vacuum) that allows them to influence each other. We model the system as a set of nodes that represent the pulse-coupled entities and a set of communication links that represent their interconnectivity.

The underlying topology considered is an arbitrary, non-partitioned digraph ranging from fully connected to 1-connected network of $K \geq 1$ nodes that exchange messages through a set of communication links. Nodes are anonymous, i.e., they do not have unique identities. All nodes are assumed to be good, i.e., actively participate in the synchronization process and correctly execute the protocol. The communication links are assumed to be between distinct nodes. All communication links are assumed to be good, i.e., reliably transfer data from their source nodes to their destination nodes. The nodes communicate with each other by exchanging broadcast messages. Broadcast of a message by a node is realized by transmitting the message, at the same time, to all nodes that are directly connected to it. The communication network does not guarantee any relative order of arrival of a broadcast message at the receiving nodes, that is, a consistent delivery order of a set of messages does not necessarily reflect the temporal or causal order of the message transmissions [Kop 1997]. There is neither a central system clock nor an externally generated global pulse or message at the network level. The communication links and nodes can behave arbitrarily provided that eventually the system adheres to the protocol assumptions (Section 3.4).

2.1. Drift Rate (ρ)

Each node is driven by an independent, free-running local physical oscillator (i.e., the phase is not controlled in any way) and a logical-time clock (i.e., a counter), denoted *LocalTimer*, which locally keeps track of the passage of time and is driven by the local physical oscillator. An **oscillator tick**, also called a **clock tick** or a **system tick**, is a discrete value and the basic unit of time in the network [Tor 2005].

An ideal oscillator has zero drift rate with respect to real-time, perfectly marking the passage of time. Real oscillators are characterized by non-zero drift rates with respect to real-time. The oscillators of the nodes are assumed to have a known bounded drift rate, ρ, which is a small constant with respect to real-time, where ρ is a unitless non-negative real value and is constrained to $0 \leq \rho << 1$. The maximum drift of the fastest *LocalTimer* over a time interval of t is given by $(1+\rho)t$. The maximum drift of the slowest *LocalTimer* over a time interval of t is given by $(1/(1+\rho))t$. Therefore, the **maximum relative drift** of the fastest and slowest nodes with respect to each other over a time interval of t is given by the following equation.

$$\delta(t) = ((1+\rho) - 1/(1+\rho))t \qquad (1)$$

Although generally ρ is bounded by $0 \leq \rho << 1$, in practice, ρ is assumed to be very close to zero. The upper bound on ρ is extensively discussed in [Mal 2011A] where we determined, at least for the protocol presented in this report, that ρ is bounded by $0 \leq \rho < 0.3$ even for theoretical purposes. Nevertheless, in the rest of this report, we'll use the generally accepted bounds for ρ, i.e., $0 \leq \rho << 1$.

2.2. The Logical Clock (*LocalTimer*)

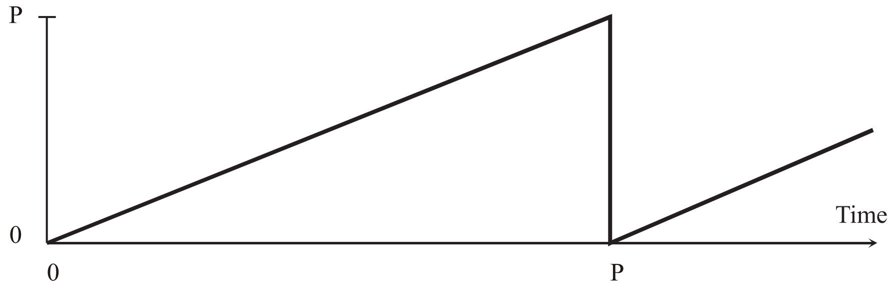

Figure 1. The *LocalTimer*.

The *LocalTimer* is driven by the local physical oscillator, takes on discrete values, and locally keeps track of the passage of time. As shown in Figure 1, the *LocalTimer* is a monotonic linear function increasing from an initial value to a maximum value. If uninterrupted, the *LocalTimer* periodically takes on all integer values from its initial value, 0, to its maximum value, P, linearly increasing within each period, thus, the *LocalTimer* is bounded by $0 \leq LocalTimer \leq P$.

2.3. Communication Delay (D, d, and γ)

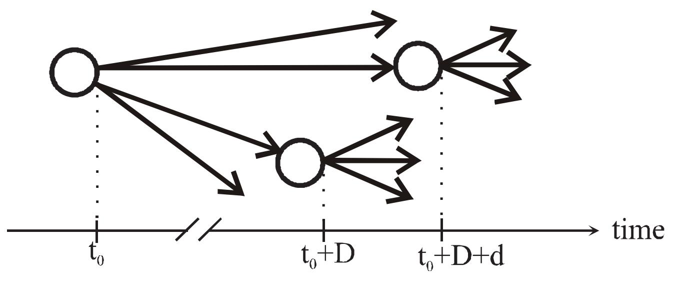

Figure 2. Event-response delay, D, and network imprecision, d.

The communication delay between adjacent 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 2. As depicted in this figure, a message transmitted by a node at real time t_0 is expected to arrive at its directly connected adjacent nodes, be processed, and subsequent messages generated by those nodes 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 receivers of a message from a transmitting node with respect to real time. The imprecision is due to the drift of the oscillators 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 units of real time clock tick. The communication latency, denoted γ, is expressed in terms of D and d, and is defined as $\gamma = (D+d)$ and so has units of real time clock ticks. Therefore, the communication delay between any two adjacent nodes is bounded by $[D, \gamma]$.

2.4. Topology (*T*)

A communication link, or simply link, is an edge in the graph representing a direct physical connection between two nodes. A path is a logical connection between two nodes consisting of one or more links. A path-length is the number of links connecting any two nodes.

The general topology, T, considered is a strongly connected directed graph (digraph) consisting of K nodes, where each node is connected to the graph by at least one link, there is a path from any node to any other node, and the links are either unidirectional or bidirectional. Furthermore, we assume there is no direct link from a node to itself, i.e., no self-loop, and there are no multiple links directly connecting any two nodes in any one direction.

In this report, we use the terms network and graph interchangeably. The following graph specific terms are used in the subsequent sections.

- Two nodes are said to be **adjacent** to each other or neighbors if they are connected to each other via a direct communication link.
- L, an integer value, is the number of links denoting the largest **loop** in the graph, i.e., the maximum value of the longest path-lengths from a node back to itself visiting the nodes along the path only once (except for the first node which is also the last node).
- W, an integer value, is the number of links signifying the **width** or diameter of the graph, i.e., the maximum value of the shortest path connecting any two nodes.

For digraphs of size $K > 1$, L and W are bounded by $2 \leq L \leq K$ and $1 \leq W \leq K - 1$.

3. The Protocol

In this section we enumerate protocol assumptions, properties, parameters, and describe the protocol in pseudo-code. The general form of the distributed synchronization problem, *S*, is defined by the following septuple [Mal 2011A].

$$S = (K, T, D, d, \rho, P, F)$$

In other words, the distributed synchronization problem is a function of the number of nodes (*K*), network topology (*T*), event-response delay (*D*), communication imprecision (*d*), oscillator drift rate (ρ), synchronization period (*P*), and number of faults (*F*), respectively. The solution to this problem is a protocol with convergence and closure properties, at a minimum, as discussed subsequently in this section. However, in the protocol presented in this report we do not deal with faults, thus $F = 0$.

Each node is driven by an independent logical-time clock, *LocalTimer*. The clocks need to be periodically synchronized due to their inherent drift with respect to each other. In order to achieve synchronization, the nodes communicate by exchanging **Sync** messages. A node is said to **time out** when its *LocalTimer* reaches its maximum value. Upon time out, the node generates a new *Sync* message and broadcasts it to others. A node is said to be **interrupted** when it accepts an incoming *Sync* message before its *LocalTimer* reaches the maximum value, i.e., before it times out. Upon interrupt and except for a predefined window (Section 3.1), the node relays the incoming *Sync* message by broadcasting it to others.

The periodic time synchronization after achieving the initial synchrony is referred to as the **resynchronization process** whereby all nodes reengage in the synchronization process. The resynchronization process begins when the first node times out and transmits a *Sync* message and ends after the last node transmits a *Sync* message. For $\rho << 1$, the fastest node cannot time out again before the slowest node transmits a *Sync* message [Mal 2011A].

A *Sync* message is transmitted either as a result of a resynchronization timeout, or when a node receives *Sync* message(s) indicative of other nodes engaging in the resynchronization process. The messages to be delivered to the destination nodes are deposited on communication links.

The following definitions and terms are used in the description and operation of the protocol. Figure 3 is used to help with the descriptions. All protocol parameters and the network level measurements are real values with time-based terms having units of real time clock ticks. However, locally and at the node level, all parameters are discrete. The discretization is for practical purposes when implementing the protocol.

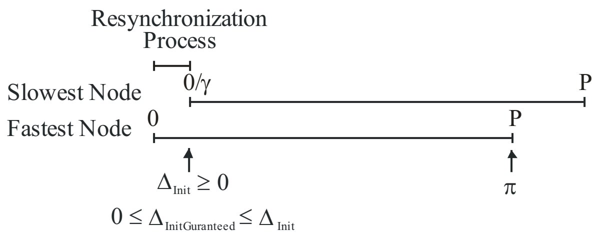
Resynchronization Process
Slowest Node
Fastest Node
0/γ
P
0
P
$\Delta_{Init} \geq 0$
π
$0 \leq \Delta_{InitGuranteed} \leq \Delta_{Init}$

Figure 3. Resynchronization process and various precisions.

- The **resynchronization period**, denoted *P*, has units of real time clock ticks and is defined as the upper bound on the time interval between any two consecutive resets of the *LocalTimer* by a node.
- **Drift per *t***, denoted $\delta(t)$, has units of real time clock ticks and is defined as the maximum amount of drift between any two nodes for the duration of *t*, $\delta(t) \geq 0$. In particular:
  - Drift per *D*, denoted $\delta(D)$, for the duration of one *D*, $\delta(D) \geq 0$.
  - Drift per γ, denoted $\delta(\gamma)$, for the duration of one γ, $\delta(\gamma) \geq 0$.
  - Drift per *P*, denoted $\delta(P)$, for the duration of one period *P*, $\delta(P) \geq 0$.
- The **graph threshold**, T_S, is based on a specified graph topology and has units of real time clock ticks.
- The guaranteed precision or simply **precision** of the network, denoted π, $0 \leq \pi < P$, has units of real time clock ticks and is defined as the guaranteed achievable precision among all nodes.
- The **convergence time**, denoted *C*, has units of real time clock ticks and is defined as the bound on the maximum time it takes for the network to converge, i.e., to achieve synchrony.
- **Precision between** *LocalTimers* of any two adjacent nodes N_i and N_j is denoted by Δ_{ij} and has units of real time clock ticks.
- The **initial synchrony** is a state of the network and the earliest time when the precision among all nodes, upon convergence, is within π. The initial synchrony occurs at time C_{Init}.
- The **initial precision** among *LocalTimers* of all nodes is denoted by Δ_{Init}, has units of real time clock ticks and, for all $t \geq C_{Init}$, is defined as a measure of the precision of the network immediately after a resynchronization process.
- The **initial guaranteed precision** among *LocalTimers* of all nodes is denoted by $\Delta_{InitGuaranteed}$, has units of real time clock ticks and, for all $t \geq C$, is defined as a measure of the precision of the network immediately after a resynchronization process.

3.1. The Graph Threshold (T_S)

When a node receives a *Sync* message, except during a predefined window, it accepts the *Sync* message, resets its *LocalTimer* and relays the *Sync* message by broadcasting it to others. The predefined window where the node ignores all incoming *Sync* messages, referred to as **ignore window**[1], provides a means for the protocol to stop the endless cycle of resynchronization triggered by the follow up *Sync* messages [Mal 2011A]. We bound the ignore window to [D, T_S). The lower bound is due to the minimum event-response delay, *D*, and the upper bound, referred to as the graph threshold, T_S, is a function of a specified graph topology.

[1] The term *refractory period* is used in biologically inspired work indicating a brief period of time, following the stimulation of a nerve, during which the nerve will not respond to a second stimulus.

3.2. Sync Message And Its Validity

In order to achieve synchrony, the nodes communicate by exchanging *Sync* messages[2]. When the system is in synchrony, the protocol overhead is at most one message per resynchronization period *P*. Assuming physical-layer error detections are dealt with separately, the reception of a *Sync* message is indicative of its validity in the value domain. The protocol performs as intended when the timing requirements of the messages from every node are satisfied. However, in the absence of faults, the reception of a *Sync* message is indicative of its validity in the value and time domains. A valid *Sync* message is discarded after it is relayed to the synchronizer (see Section 3.3) and has been kept for one local clock tick.

3.3. The Monitor, The Synchronizer, And Protocol Functions

A node consists of a **synchronizer** and a set of **monitors**. To assess the behavior of other nodes, a node employs as many monitors as the number of nodes it is directly connected to with one monitor for each source of incoming messages. A node neither uses nor monitors its own messages. A monitor keeps track of the activities of its corresponding source node. Specifically, a monitor reads, evaluates, validates, and stores the last valid message it receives from that node. Upon conveying the valid message to the local synchronizer, a monitor disposes of the valid message after it has been kept for one local clock tick. The assessment results of the monitored nodes are utilized by the synchronizer in the synchronization process.

The function *ValidateMessage()*, Figure 4, used by the monitors determines whether a received *Sync* message is valid. We assume physical-layer error detections are dealt with separately. The function *ConsumeMessage()* used by the monitors invalidates the stored *Sync* message after it has been kept for one local clock tick. The function *ValidSync()* used by the synchronizer examines availability of valid *Sync* messages.

```
ValidateMessage():
if (incoming message =  Sync) then
{Message is valid,
  Store it.}

ConsumeMessage():
if (stored message timer ≥ 1 tick) then
{Message is expired,
  Clear it.}

ValidSync():
if (number of stored messages > 0) then
{    return true,
  else
      return false.}
```

Figure 4. The protocol functions.

[2] Since only one message type is used for the operation of this protocol, a single bit suffices.

3.4. Protocol Assumptions

1. $K \geq 1$.
2. All nodes correctly execute the protocol.
3. All links correctly transmit data from their sources to their destinations.
4. T is a non-partitioned, strongly connected digraph.
5. $0 \leq \rho << 1$.
6. A message sent by a node will be received and processed by its adjacent nodes within γ, where $\gamma = (D + d)$.
7. The initial values of the variables of a node are within their corresponding data-type range, although possibly with arbitrary values. (In an implementation, it is expected that some local mechanism exists to enforce type consistency for all variables.)

3.5. The Self-Stabilizing Distributed 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 $t_0 = 0$ and for all $t \geq t_0$ the system operates within the *protocol assumptions*. The maximum difference in the value of *LocalTimer* for all pairs of nodes at time t, $\Delta_{Net}(t)$, is determined by the following equation that accounts for the variations in the values of the *LocalTimer* across all nodes.

$$r = \lceil (W + 1)(\gamma + \delta(\gamma)) \rceil,$$
$$LocalTimer_{min}(x) = min\ (N_i.LocalTimer(x)), \text{ and}$$
$$LocalTimer_{max}(x) = max\ (N_i.LocalTimer(x)), \text{ for all } i.$$
$$\Delta_{Net}(t) = min\ ((LocalTimer_{max}(t) - LocalTimer_{min}(t)),\ (LocalTimer_{max}(t - r) - LocalTimer_{min}(t - r))).$$

The following symbols were defined earlier and are listed here for reference:

- P denotes the resynchronization period, has units of real time clock ticks, and is defined as the upper bound on the time interval between any two consecutive resets of the *LocalTimer* by a node and $P > 0$.
- C denotes a bound on the maximum convergence time.
- $\Delta_{Net}(t)$, for real time t, is the maximum difference of values of the *LocalTimers* of any two nodes (i.e., the relative clock skew) for $t \geq t_0$.
- π, the synchronization precision, is the guaranteed upper bound on $\Delta_{Net}(t)$ for all $t \geq C$, and is generally assumed to be very small compared to P.

To show that a protocol is self-stabilizing, it has to be proven that there exist C and π such that the following self-stabilization properties hold.

1. **Convergence:** $\Delta_{Net}(C) \leq \pi$, $0 \leq \pi < P$
2. **Closure:** For all $t \geq C$, $\Delta_{Net}(t) \leq \pi$
3. **Congruence:** For all nodes N_i, for all $t \geq C$, ($N_i.LocalTimer(t) = \gamma$) implies $\Delta_{Net}(t) \leq \pi$.
4. **Liveness:** For all $t \geq C$, *LocalTimer* of every node sequentially takes on at least all integer values in $[\gamma, P - \pi]$.

3.6. The Self-Stabilizing Distributed Clock Synchronization Protocol For Arbitrary Digraphs

The protocol, executed by all nodes, is presented in Figure 5 and consists of a synchronizer and a set of monitors which execute once every local clock tick. The *if* statement describing the synchronizer has five parts that are labeled *E0* through *E4* and referenced subsequently in this report.

```
Synchronizer:
E0: if ($LocalTimer < 0$)
        $LocalTimer := 0$,
E1: elseif ($ValidSync()$ and ($LocalTimer < D$))
        $LocalTimer := \gamma$,        // interrupted
E2: elseif (($ValidSync()$ and ($LocalTimer \geq T_S$))
        $LocalTimer := \gamma$,        // interrupted
        Transmit Sync,
E3: elseif ($LocalTimer \geq P$)      // timed out
        $LocalTimer := 0$,
        Transmit Sync,
E4: else
        $LocalTimer := LocalTimer + 1$.
```

```
Monitor:
case (message from the corresponding node)
{Sync:
        ValidateMessage()
 Other:
        Do nothing.
} // case
ConsumeMessage()
```

Figure 5. The self-stabilizing clock synchronization protocol for arbitrary digraphs[3].

The following is a list of protocol parameters when all links are bidirectional.

$T_S \geq (L+2)(\gamma + \delta(\gamma))$

$P \geq 3T_S$, for $\rho = 0$

$P \geq 3(T_S + \delta(T_S))$, for $L = K$ and $\rho > 0$

$P \geq \max((2K + 1)(\gamma + \delta(\gamma)), 3(T_S + \delta(T_S)))$, for $L = f(T)$ and $\rho > 0$

The following is a list of protocol parameters for digraphs, i.e., when at least one link is unidirectional.

$T_S \geq (K+2)(\gamma + \delta(\gamma))$

$P \geq K(T_S + \delta(T_S))$

Regardless of the types of links in the network, the following is a list of protocol measures.

$C_{Init} = 2P + K(\gamma + \delta(\gamma))$

$\Delta_{Init} \leq (K - 1)(\gamma + \delta(\gamma))$

$C = C_{Init} + \lceil \Delta_{Init} / \gamma \rceil P$

$0 \leq \Delta_{InitGuaranteed} \leq W(\gamma + \delta(\gamma))$, for all $t \geq C$

$\pi = \Delta_{InitGuaranteed} + \delta(P)$, $0 \leq \pi < P$, for all $t \geq C$

A trivial solution is when $P = 0$. Since $P > T_S$, $T_S > 0$, and the *LocalTimer* is reset after reaching *P* (worst-case wraparound), a trivial solution is not possible.

[3] Statement *E0* makes explicit the assumption 7 in Section 3.4 (Protocol Assumptions) as it applies to the *LocalTimer*.

4. A Deductive Proof Of The Correctness Of The Protocol

In this section we present a deductive proof of the correctness of the protocol for the general case: a realizable system independent of the network topology where $\rho \geq 0$ and $d \geq 0$. Defining T_S in terms of L requires knowledge of the topology of the given network. Therefore, in order to generalize the expression for T_S, make it independent of the topology, and to help simplify the proof process, we express it in terms of its worst case value, $L = K$, which implies that $T_S \geq (K+2)(\gamma + \delta(\gamma))$. However, for a specific application, optimizing T_S by expressing it in terms of L results in faster synchrony and better performance.

The deductive proof presented in this section is for the protocol presented in previous section (Section 3.6) and reported earlier in [Mal 2011A]. We do not provide separate proofs for the variations of the protocol as were discussed in [Mal 2011A]. We believe however that this proof can readily be extended to all variations of the protocol.

The proof idea is depicted in Figure 6. The main theorems address the following questions. Assuming a *Sync* message does not get ignored and *P* is sufficiently large, is it possible for a message to circulate within the network without dying out? In other words, will *E2* (Figure 5)[4] get executed indefinitely? Is it possible for a node to transmit *Sync* messages without ever timing out? In other words, will *E3* ever get executed? Also, will *E4* ever get executed?

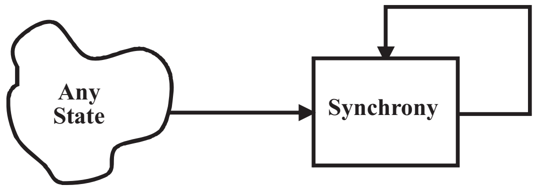

Figure 6. Proof approach.

The ranges of the protocol parameters used in the proofs are restated here for reference.

$K \geq 2$ ($K = 1$ is a simple case and does not need a proof.)
$L = K$
$W = K - 1$
$0 \leq \rho << 1$
$1 \leq D \leq \gamma < T_S < P$
$0 \leq LocalTimer \leq P$
Ignore window = $[D, T_S)$
$T_S \geq (K+2)(\gamma + \delta(\gamma))$
$P \geq K(T_S + \delta(T_S))$

We'd like to emphasize that *P* is defined as the upper bound on the time interval between any two consecutive resets by any node in the network, thus, its value is specified with respect to the slowest node. In other words, assuming the fastest node and the slowest node start at initial synchrony, when the fastest node reaches *P*, the slowest node is at $P - \delta(P)$.

[4] Labels *E0* through *E4* of Figure 5 refer to different parts of the synchronizer.

Lemma *JoinTimedOut – When a node, N_j, is interrupted by an adjacent node that was timed out, N_i, it synchronizes with that node with a relative initial precision of $\Delta_{ij} = d + \delta(D)$.*
Proof – Given N_j was interrupted at time t by an incoming *Sync* message from N_i, by the protocol (part *E1*, *E2*), N_j sets its *LocalTimer* while accounting for the message arrival delay, i.e., $N_j.LocalTimer = \gamma$. Given N_i had timed out (part *E3* of the protocol), it must have reset its *LocalTimer* to 0 and sent the *Sync* message between D and γ time units earlier. Accounting for drift, at time t,

$N_i.LocalTimer$ = 0 + actual communication delay + relative drift to N_j.

The communication delay is bounded by $[D, \gamma]$. Thus, with N_i as either the slower node or the faster node than N_j,

$D - \delta(D) \leq N_i.LocalTimer \leq \gamma + \delta(\gamma)$.

Since the relative precision is an upper bound,

$\Delta_{ij} = \max(\text{abs}(N_j.LocalTimer - N_i.LocalTimer))$.

So,

$\Delta_{ij} = (\gamma + \delta(\gamma)) - \gamma = \delta(\gamma)$, and
$\Delta_{ij} = \gamma - (D - \delta(D)) = d + \delta(D)$.

Since $\delta(\gamma) = \delta(D) + \delta(d)$ and $0 \leq \rho << 1$, then $d \geq \delta(d)$, and therefore,

$\Delta_{ij} = \max(\delta(\gamma), d + \delta(D))$
$\Delta_{ij} = d + \delta(D)$. ■

Lemma *JoinInterrupted – When a node, N_j, is interrupted by an adjacent node that was in turn interrupted, N_i, it synchronizes with that node with a relative initial precision of $\Delta_{ij} = \gamma + \delta(\gamma)$.*
Proof – Given N_j was interrupted at time t by an incoming *Sync* message from N_i, by the protocol (parts *E1*, *E2*), N_j sets its *LocalTimer* while accounting for the message arrival delay, i.e., $N_j.LocalTimer = \gamma$. Given N_i was interrupted (parts *E1*, *E2* of the protocol), it must have set its *LocalTimer* to γ and sent the *Sync* message between D and γ time units earlier. Accounting for drift, at time t,

$N_i.LocalTimer = \gamma$ + actual communication delay + relative drift to N_j.

The communication delay is bounded by $[D, \gamma]$. Thus, with N_i as either the slower node or the faster node than N_j,

$\gamma + D - \delta(D) \leq N_i.LocalTimer \leq \gamma + \gamma + \delta(\gamma)$.

Since the relative precision is an upper bound,

$\Delta_{ij} = \max(\text{abs}(N_j.LocalTimer - N_i.LocalTimer))$.

So,

$\Delta_{ij} = (2\gamma + \delta(\gamma)) - \gamma = \gamma + \delta(\gamma)$, and
$\Delta_{ij} = (\gamma + D - \delta(D)) - \gamma = D - \delta(D)$.

Since $\gamma \geq D$,

$\Delta_{ij} = \max (\gamma + \delta(\gamma), D - \delta(D))$

$\Delta_{ij} = \gamma + \delta(\gamma)$. ■

We have defined the ignore window earlier as the predefined time interval (window) where the node ignores all incoming *Sync* messages as a means for the protocol to stop the endless cycle of resynchronization triggered by the subsequent *Sync* messages. From the protocol (parts *E1*, *E2*), the ignore window for N_i is the time interval of $D \leq N_i.LocalTimer < T_S$.

Lemma *IgnoredNode – When a node, N_j, ignores a Sync message from an adjacent node, N_i, the two nodes have a relative initial precision of $\Delta_{ij} < T_S - D + \delta(D)$.*

Proof – Given, at time *t*, N_j ignored an incoming *Sync* message from N_i, by the protocol, then $D \leq N_j.LocalTimer < T_S$. Given N_i had sent the *Sync* message between *D* and γ time units earlier, by the protocol (parts *E2*, *E3*), N_i was either interrupted or had timed out. We address these cases separately.

Case 1 – Given N_i was interrupted, it must have set its *LocalTimer* to γ and sent the *Sync* message between *D* and γ time units earlier (part *E2* of the protocol). Accounting for drift, at time *t*,

$N_i.LocalTimer = \gamma +$ actual communication delay + relative drift to N_j.

The communication delay is bounded by $[D, \gamma]$. Thus, with N_i as either the slower node or the faster node than N_j,

$$\gamma + D - \delta(D) \leq N_i.LocalTimer \leq \gamma + \gamma + \delta(\gamma). \qquad (1)$$

Since the relative precision is an upper bound,

$\Delta_{ij} = \max (\text{abs} (N_j.LocalTimer - N_i.LocalTimer))$

$\Delta_{ij} < T_S - (\gamma + D - \delta(D))$.

Case 2 – Given N_i had timed out, it must have reset its *LocalTimer* to 0 and sent the *Sync* message between *D* and γ time units earlier (part *E3* of the protocol). Accounting for drift, at time *t*,

$N_i.LocalTimer = 0 +$ actual communication delay + relative drift to N_j.

The communication delay is bounded by $[D, \gamma]$. Thus, with N_i as either the slower node or the faster node than N_j,

$$D - \delta(D) \leq N_i.LocalTimer \leq \gamma + \delta(\gamma). \qquad (2)$$

Since the relative precision is an upper bound,

$\Delta_{ij} = \max (\text{abs} (N_j.LocalTimer - N_i.LocalTimer))$

$\Delta_{ij} < T_S - (D - \delta(D))$.

Since the relative precision is an upper bound, from the above cases, $\Delta_{ij} < T_S - D + \delta(D)$. ■

Lemma *JoinTimedOutAfterIgnored – When all adjacent nodes are within a relative precision of $\Delta_{ij} < T_S - D+\delta(D)$ clock ticks of each other, for any two adjacent nodes N_i and N_j with $N_i.LocalTimer < N_j.LocalTimer$, if N_j times out, N_i will join N_j with a relative initial precision of $\Delta_{ij} = d + \delta(D)$.*
Proof – Given at time t the relative precision of Δ_{ij} and N_i lagging N_j by Δ_{ij}, since $0 \leq \rho << 1$, $D >> \delta(D)$, $\Delta_{ij} < T_S$, and so $P - \Delta_{ij} > P - T_S$.

$P \geq KT_S + K\delta(T_S)$, subtracting T_S from both sides,
$P - T_S \geq (K-1)T_S + K\delta(T_S)$, and so,
$P - \Delta_{ij} > (K-1)T_S + K\delta(T_S)$.

At time $t' = t + (P - \Delta_{ij})$, when N_j times out, $N_i.LocalTimer$ is determined as follows. For the worst case analysis, we assume N_i is the slower node than N_j. Also, even though $P - \Delta_{ij} < P$, i.e., $\delta(P - \Delta_{ij}) < \delta(P)$, we set $\delta(P - \Delta_{ij}) = \delta(P)$ to simplify the algebraic argument.
$N_i.LocalTimer(t') = N_i.LocalTimer(t) + (P - \Delta_{ij}) - \delta(P - \Delta_{ij})$
$N_i.LocalTimer(t') = N_i.LocalTimer(t) + ((K-1)T_S + K\delta(T_S)) - \delta(P)$

Since $\delta(P) = K\delta(T_S)$,
$N_i.LocalTimer(t') = N_i.LocalTimer(t) + (K-1)T_S$

From inequalities (1) and (2) of the proof of Lemma *IgnoredNode*, and since we consider for the worst case analysis N_i being the slower node than N_j,
$N_i.LocalTimer(t) = D - \delta(D)$, and
$N_i.LocalTimer(t') \geq D - \delta(D) + (K-1)T_S$.

Since $0 \leq \rho << 1$, $D >> \delta(D)$, and for all $K \geq 2$,
$N_i.LocalTimer(t') > T_S$.

Thus, when N_j times out, $N_i.LocalTimer > T_S$ and N_i accepts N_j's message. By the Lemma *JoinTimedOut* the two adjacent nodes N_i and N_j synchronize with each other with a relative initial precision of $\Delta_{ij} = d + \delta(D)$ clock ticks. ■

Lemma *JoinInterruptedAfterIgnored1 – When at time t all adjacent nodes are within a relative precision of $\Delta_{ij} < T_S - D+\delta(D)$ clock ticks of each other, for any two adjacent nodes N_i and N_j with $N_i.LocalTimer < N_j.LocalTimer$, if N_j gets interrupted by another node at $t' \geq t+T_S+\delta(T_S)$, N_i will join N_j with a relative initial precision of $\Delta_{ij} = \gamma + \delta(\gamma)$.*
Proof – Given the relative precision of Δ_{ij} and N_i lagging N_j by Δ_{ij}, at time t, for the worst case analysis, we assume N_i is the slower node than N_j. At time $t' = t + T_S + \delta(T_S)$,
$N_j.LocalTimer(t') = N_j.LocalTimer(t) + T_S + \delta(T_S)$
$N_i.LocalTimer(t') = N_j.LocalTimer(t') - \delta(T_S)$, and so,
$N_i.LocalTimer(t') = N_i.LocalTimer(t) + T_S$

From inequalities (1) and (2) of the proof of Lemma *IgnoredNode*, and since we consider for the worst case analysis N_i being the slower node than N_j,
$N_i.LocalTimer(t) = D - \delta(D)$, and

$N_i.LocalTimer(t') \geq D - \delta(D) + T_S$.

Since $0 \leq \rho << 1$, $D >> \delta(D)$, therefore,
$N_i.LocalTimer(t') > T_S$.

Thus, when N_j times out, $N_i.LocalTimer > T_S$ and N_i accepts N_j's message. By the Lemma *JoinInterrupted* the two adjacent nodes N_i and N_j synchronize with each other with a relative initial precision of $\Delta_{ij} = \gamma + \delta(\gamma)$ clock ticks. ■

Lemma *JoinInterruptedAfterIgnored2 – When all adjacent nodes are within a relative precision of* $\Delta_{ij} < T_S - D+\delta(D)$ *clock ticks of each other, for any two adjacent nodes* N_i *and* N_j *with* $N_i.LocalTimer < N_j.LocalTimer$, *if* N_i *gets interrupted by another node before either* N_j *times out or gets interrupted by yet another node,* N_j *will join* N_i *with a relative initial precision of* $\Delta_{ij} = \gamma + \delta(\gamma)$.
Proof – Given the relative precision of Δ_{ij} and N_i lagging N_j by Δ_{ij} and N_i gets interrupted by another node (other than N_j), by the protocol (part *E3*), $T_S \leq N_i.LocalTimer < P$, i.e., N_i had exited the ignore window. Since N_i lags N_j, $T_S \leq N_j.LocalTimer \leq P$. By the Lemma *JoinInterrupted* the two adjacent nodes N_i and N_j synchronize with each other with a relative initial precision of $\Delta_{ij} = \gamma + \delta(\gamma)$ clock ticks. ■

Note in the general form of the above lemma if either of the two nodes times out, the other one will follow and $\Delta_{ij} = d + \delta(D)$. However, since $N_i.LocalTimer < N_j.LocalTimer$, if N_i times out before N_j, it implies that N_i is a considerably faster node and the amount of drift in the system is very large; $\delta(P) > T_S$.

It follows from the protocol that an incoming *Sync* message to a node either gets ignored or gets accepted and subsequently relayed to other nodes. A *Sync* message is said to **die out** when all receiving nodes ignore it. Assuming the message does not get ignored and *P* is sufficiently large, is it possible for a *Sync* message to circulate within the network without ever dying out? In other words, will *E2* get executed indefinitely? Is it possible for a node to transmit *Sync* messages without ever timing out? In other words, will *E3* ever get executed? How about *E4*? The following lemmas address these questions.

Lemma *NoInfiniteLoop – A Sync message always dies out.*
Proof – By the protocol (parts *E1*, *E2*) a message dies out if it gets ignored by all receiving nodes ($D \leq N_i.LocalTimer < T_S$), otherwise, it gets relayed to other nodes. For a message to persist it has to circulate within the network forever, i.e., there needs to be a loop in the network. By definition, *L* signifies the size of the largest loop in the network. Let all nodes in a loop accept a *Sync* message when they receive it, relay the incoming *Sync* message, and no new messages are generated due to a time out. Let, at time *t*, N_i be the first node in that loop that just got interrupted by another node. So, $N_i.LocalTimer = \gamma$ and N_i will relay the *Sync* message to other nodes inside the loop. At time *t'*, this message will make it back to N_i after traversing the loop. Accounting for drift and assuming N_i is the fastest node, at time *t'*,
$N_i.LocalTimer = \gamma + L\gamma + L\ \delta(\gamma)$,
$N_i.LocalTimer = (L + 1)\gamma + L\ \delta(\gamma)$.

By the protocol (parts *E1*, *E2*), N_i will ignore the message if $D \le N_i.LocalTimer < T_S$. At time *t'* for N_i,

$D \le (L + 1)\gamma + L\,\delta(\gamma)$, since $L \ge 1$, $\gamma \ge D$, and $L\,\delta(\gamma) \ge 0$, and
$(L + 1)\gamma + L\,\delta(\gamma) < T_S$
$(L + 1)\gamma + L\,\delta(\gamma) < (K + 2)\gamma + (K+2)\,\delta(\gamma)$.

Since $L \le K$, substituting K for L in the above inequality, we get the following.

$(K + 1)\gamma + \delta(\gamma)K < (K + 2)\gamma + (K+2)\,\delta(\gamma)$
$0 < \gamma + 2\delta(\gamma)$.

Since $\gamma \ge 1$ and $2\delta(\gamma) \ge 0$, the above inequality holds true. Thus, $D \le N_i.LocalTimer < T_S$ at time *t'* and N_i will ignore the message. Therefore, the message will die out even if it reaches its original source. As a result, it is impossible for a message to persist within the network indefinitely. ■

Lemma *MessageLifeSpan – A Sync message dies out in at most $K(\gamma + \delta(\gamma))$ clock ticks.*
Proof – By the Lemma *NoInfiniteLoop*, a *Sync* message always dies out. It also follows that if there is a loop in the graph, it takes at most $L(\gamma + \delta(\gamma))$ clock ticks for a message to reach its source and then die out. By the definition of W, it takes at most $W(\gamma + \delta(\gamma))$ clock ticks for a message to reach all other nodes. Once again, by the Lemma *NoInfiniteLoop*, this *Sync* message always dies out. Since $L \le K$ and $W \le K$-1, for the worst case analysis we choose the largest value for L and W, i.e., K. Thus, it takes at most $K(\gamma+\delta(\gamma))$ clock ticks for a *Sync* message to die out. ■

Lemma *IncLocalTimer – The LocalTimer of at least one node always reaches P.*
Proof – By the Lemma *NoInfiniteLoop* a *Sync* message always dies out. Also, by the Lemma *NoInfiniteLoop* unless new *Sync* messages are generated, all existing *Sync* messages in the network will eventually die out. In the absence of a *Sync* message circulating within the network, by the protocol (part *E4*), the *LocalTimer* gets incremented until it reaches its maximum value P. Therefore, the *LocalTimer* of at least one node always reaches P. ■

Lemma *NewSync – Within every time interval of P clock ticks, at least one node generates a new Sync message.*
Proof – By the Lemma *NoInfiniteLoop* a *Sync* message always dies out. By the Lemma *MessageLifeSpan* a *Sync* message dies out in $K(\gamma+\delta(\gamma))$ clock ticks. Since $K(\gamma+\delta(\gamma)) < P$, by the Lemma *IncLocalTimer* at least one node reaches P. By the protocol (part *E3*), once a node reaches P, it times out and generates a new *Sync* message. Therefore, within every P at least one node always times out and generates a new *Sync* message. ■

Lemma *AllNodesTxSync – Within every time interval of P clock ticks, every node N_i transmits at least one Sync message.*
Proof – We prove this lemma by examining all activities along the time line.

By the protocol (part *E1*), when $N_i.LocalTimer < D$, if N_i does not receive a *Sync* message, its *LocalTimer* gets incremented (part *E4*), reaches D, and the node enters the ignore window. However, if N_i receives a *Sync* message, it accepts the message without relaying it. In this case, the node sets its *LocalTimer* to γ and since $\gamma \geq D$ this event occurs only once and the node enters the ignore window.

By the protocol while in the ignore window (parts *E1*, *E2*), $D \leq N_i.LocalTimer < T_S$, the node rejects all incoming *Sync* messages and the *LocalTimer* gets incremented (part *E4*), reaches T_S, and the node exits the ignore window.

By the protocol after exiting the ignore window (part *E3*), $T_S \leq N_i.LocalTimer \leq P$, if N_i receives a *Sync* message from another node, it will accept and relay it, otherwise, its *LocalTimer* gets incremented (part *E4*), reaches its maximum value *P*, and the node times out, generates a new *Sync* message, and transmits it to others.

Therefore, within any *P* clock ticks, every node N_i transmits at least one *Sync* message. ■

Lemma *DeltaijLessThanTs – For all* $t \geq P+\gamma$ *clock ticks the relative initial precision of any two adjacent nodes* N_i *and* N_j *is* $\Delta_{ij} < T_S - D+\delta(D)$.
Proof – By the Lemma *AllNodesTxSync*, within every time interval of *P* clock ticks every node transmits at least one *Sync* message. Accounting for the message processing and delivery time, i.e., γ, by the Lemmas *JoinTimedOut*, *JoinInterrupted*, and *IgnoredNode*, the relative initial precision of any two adjacent nodes N_i and N_j within $P+\gamma$, is $d + \delta(D)$, $\gamma + \delta(\gamma)$, or less than $T_S - D+\delta(D)$ clock ticks, respectively. Since the relative precision is an upper bound, therefore, $\Delta_{ij} < T_S - D+\delta(D)$. ■

Lemma *PrecisionLessThanWTs – For all* $t \geq P+\gamma$ *clock ticks the initial network precision is* $\Delta_{Net}(t) < W(T_S - D+\delta(D))$.
Proof – By the Lemma *DeltaijLessThanTs*, for all $t \geq P+\gamma$ clock ticks the relative initial precision of any two adjacent nodes is $\Delta_{ij} < T_S - D+\delta(D)$. Since the initial precision of the network is an upper bound, by the definition of *W*, at $t = P+\gamma$, $\Delta_{Net}(t)$ is the sum of all Δ_{ij}, i.e., $\Delta_{Net}(t) = W\Delta_{ij}$. Thus, $\Delta_{Net}(t) < W(T_S - D+\delta(D))$. ■

Lemma *DeltaijAboutGamma – For all* $t \geq 2P+\gamma$ *clock ticks and upon the subsequent resynchronization process, the relative initial precision of any two adjacent nodes* N_i *and* N_j *is* $\Delta_{ij} = \gamma +\delta(\gamma)$.
Proof – By the Lemma *PrecisionLessThanWTs*, for all $t \geq P+\gamma$ clock ticks the initial network precision is $\Delta_{Net}(t) < W(T_S - D+\delta(D))$. Within another *P*, i.e., at $t = 2P+\gamma$, the $\Delta_{Net}(t)$ increases by the maximum drift for the duration of *P*, i.e., $\delta(P)$. So,

$$\Delta_{Net}(t) < W(T_S - D+\delta(D)) + \delta(P).$$

With $W \leq K - 1$, we set $W = K - 1$ for the worst case analysis, and since $\delta(P) = K\delta(T_S)$,

$$\Delta_{Net}(t) < (K-1)(T_S - D+\delta(D)) + K\delta(T_S).$$

Since $P \geq KT_S + K\delta(T_S)$,

$$P - \Delta_{Net}(t) \geq (KT_S + K\delta(T_S)) - ((K-1)(T_S -D+\delta(D)) + K\delta(T_S)), \text{ and so,}$$

$$P - \Delta_{Net}(t) \geq T_S + (K-1)(D - \delta(D)).$$

Since $0 \leq \rho << 1$, $D >> \delta(D)$ and $P - \Delta_{Net}(t) > T_S + (K\text{-}1)D$.

Thus, at $t = 2P+\gamma$, $P - \Delta_{Net}(t) > T_S$. By the Lemma *NoInfiniteLoop* a *Sync* message always dies out. By the Lemma *IncLocalTimer*, during every P time interval, at least one node reaches P and by the Lemma *NewSync*, a new *Sync* message always gets generated. At $t \geq 2P+\gamma$ and upon the subsequent resynchronization process, by the Lemmas *JoinTimedOutAfterIgnored*, *JoinInterruptedAfterIgnored1*, and *JoinInterruptedAfterIgnored2* all nodes synchronize with their adjacent nodes with relative initial precision of any two adjacent nodes as $d + \delta(D)$, $\gamma + \delta(\gamma)$, and $\gamma + \delta(\gamma)$ clock ticks, respectively. Since the initial precision is an upper bound, thus $\Delta_{ij} = \gamma + \delta(\gamma)$. ■

Lemma *InitialPrecision – For all* $t \geq C_{Init}$, *where* $C_{Init} = 2P+K(\gamma+\delta(\gamma))$ *clock ticks, and upon the subsequent resynchronization process, the initial network precision is* $\Delta_{Init} = (K\text{-}1)(\gamma+\delta(\gamma))$.
Proof – By the Lemma *DeltaijAboutGamma*, for all $t \geq 2P+\gamma$ and upon the subsequent resynchronization process, $\Delta_{ij} = \gamma + \delta(\gamma)$. By the definition of W, it takes at most $W(\gamma+\delta(\gamma))$ clock ticks for a message to reach all other nodes. Since $W \leq K\text{-}1$, for the worst case analysis we set $W = K\text{-}1$. Since the initial precision of the network is an upper bound, $\Delta_{Net}(t)$ is the sum of all Δ_{ij}, i.e., $\Delta_{Net}(t) = W\Delta_{ij} = (K\text{-}1)(\gamma+\delta(\gamma))$. Therefore, $\Delta_{Net}(t) = (K\text{-}1)(\gamma + \delta(\gamma))$, and so, $\Delta_{init} = (K\text{-}1)(\gamma + \delta(\gamma))$. ■

Theorem *InitialConvergence – For all* $t \geq C_{Init}$, *the network converges to a state where the guaranteed network precision is* $\pi = \Delta_{Init} + \delta(P)$.
Proof – By the Lemma *DeltaijAboutGamma*, for all $t \geq 2P+\gamma$ and upon the subsequent resynchronization process, $\Delta_{ij} = \gamma + \delta(\gamma)$. By the Lemma *InitialPrecision*, for all $t \geq C_{Init}$ and upon the completion of the subsequent resynchronization process, $\Delta_{Init} = (K\text{-}1)(\gamma + \delta(\gamma))$. However, due to drift, Δ_{Init} will increase over a time interval of P by a maximum amount of $\delta(P)$. Therefore, for all $t \geq C_{Init}$, the guaranteed precision of the network is

$\pi = \Delta_{Init}$ + maximum drift in the network over P

$\pi = \Delta_{Init} + \delta(P)$.

Thus, for all $t \geq C_{Init}$, the network converges with $\pi = \Delta_{Init} + \delta(P)$ and remains within π. ■

Corollary *InitialConvergenceTime – The initial convergence time is* C_{Init}.
Proof – By the Theorem *InitialConvergence* the network converges within C_{Init}. ■

Corollary *InitGuaranteedPrecision – For all* $t \geq C_{Init}$ *and upon the completion of the subsequent resynchronization process, the initial guaranteed precision of the network is* $\Delta_{InitGuaranteed} = \Delta_{Init}$.
Proof – By the Theorem *InitialConvergence* the network converges within C_{Init} with an initial precision of Δ_{Init}. It also follows that upon the completion of the subsequent resynchronization processes the initial guaranteed precision of the network for all $t \geq C_{Init}$ is $\Delta_{InitGuaranteed} = \Delta_{Init}$. ■

Theorem *InitialClosure – For all $t \geq C_{Init}$, a synchronized network where all nodes have converged to $\pi = \Delta_{Init} + \delta(P)$, shall remain within the synchronization precision π.*
Proof – By the Lemma *InitialPrecision*, for $t \geq C_{Init}$, $\Delta_{Init} = (K\text{-}1)(\gamma + \delta(\gamma))$. By the Theorem *InitialConvergence* for $t \geq C_{Init}$ the network has converged with guaranteed network precision $\pi = \Delta_{Init} + \delta(P)$ while accounting for the maximum drift over a time interval of P. By the Theorem *InitialConvergence* and upon subsequent resynchronization processes, the network precision will remain within π. Thus, the network has converged, is synchronized, and remains synchronized within π. ■

Theorem *InitialNetworkPrecision – For all $t \geq C_{Init}$, where $C_{Init} = 2P+K(\gamma+\delta(\gamma))$ clock ticks, the network precision is $\pi = \Delta_{Init} + \delta(P)$.*
Proof – By the Lemma *InitialConvergence* and Theorem *InitialClosure*, for all $t \geq C_{Init}$, $\pi = \Delta_{Init} + \delta(P)$. ■

Theorem *InitialCongruence – For all nodes N_i and for all $t \geq C_{Init}$, (N_i.LocalTimer(t) = γ) implies $\Delta_{Net}(t) \leq \pi$.*
Proof – By the protocol (part *E3*) upon time out a node resets its *LocalTimer* to zero. By the Lemma *IncLocalTimer* the *LocalTimer* of a node will always get incremented and unless interrupted a timed out node will reach γ. By the protocol (parts *E1*, *E2*) when a node gets interrupted, it resets its *LocalTimer* to γ. Thus, a node will always reach γ. By the Theorems *InitialConvergence* and *InitialClosure*, for all $t \geq C_{Init}$, the network convergences and remains synchronized with $\pi = \Delta_{Init} + \delta(P)$. Therefore, for all $t \geq C_{Init}$ and $0 \leq \rho << 1$, when N_i*.LocalTimer(t)* = γ, the network precision remains within π, i.e., $\Delta_{Net}(t) \leq \pi$, and all nodes are in synchrony. ■

The proof presented thus far is based on very conservative measures and has demonstrated the correctness of the protocol even when the network parameters are considered at their boundary (minimum and maximum) values. However, upon the initial convergence and closure, there are a number of ways to achieve tighter precision. One such method is introduced in [Mal 2006B], where, in a two step process, another proven correct protocol that is based on the initial synchrony assumptions is used to achieve the optimum precision of the coarsely synchronized system. Another method is by adding a few random links or rewiring links with a certain probability to provide shortcuts between different segments of a graph [Wat 1998][Gad 2000][Bar 2002][Hon 2002, 2004][Li 2004][Gom 2007]. These ideas, as discussed in [Mal 2011A], are primarily used to achieve convergence, but can also be used to achieve tighter precision. There is yet another method, presented below, that simply requires more time. Performance of this method depends primarily on the drift rate.

From the expression for $\Delta_{Init} = (K\text{-}1)(\gamma + \delta(\gamma))$ it is evident that the synchronization time, C, and precision, π, are also functions of the graph topology and the drift rate, specifically, the graph's width and the amount of drift the network experiences. In other words,

$C = f(T, \rho) = f(W, \delta(P))$, and

$\pi = f(T, \rho) = f(W, \delta(P))$.

Note that the general equation for Δ_{Init} encompasses the ideal and semi-ideal scenarios. In particular, for the ideal scenario, where $\rho = 0$ and $d = 0$, and for the semi-ideal scenario, where $\rho = 0$ and $d \geq 0$, $\Delta_{Init} = (K\text{-}1)\gamma$.

Thus far and in the proof process, we assumed $0 \leq \rho << 1$. The proof, however, holds when this bound is less restricted, i.e., $0 \leq \rho < 1$. When $\rho >> 0$ such that $\delta(\gamma) \geq \gamma$, by the Theorem *InitialConvergence*, for all $t \geq C_{Init}$ and upon subsequent resynchronization processes, the network converges with $\pi = \Delta_{Init} + \delta(P)$. In other words, due to the high rate of drift, no further improvement on Δ_{Init} and the network precision, π, can be guaranteed. Thus, when $\rho >> 0$, $\Delta_{InitGuaranteed} \leq \Delta_{Init}$, and no further improvement on Δ_{Init} is achieved; therefore, no improvement on π can be guaranteed. Furthermore, the convergence time C is bounded by C_{Init}, i.e., $C = C_{Init}$.

However, if $\rho << 1$, although the initial (coarse) synchrony, Δ_{Init}, occurs within C_{Init}, the initial guaranteed synchrony, $\Delta_{InitGuaranteed}$, takes place after a number of periods and upon achieving the initial synchrony, i.e., Δ_{Init}. We demonstrate this by the following lemmas for the semi-ideal scenarios where $\rho = 0$, $d \geq 0$, and $\gamma = D + d$.

Lemma *ρdJoinTimedOut – For $\rho = 0$, when a node, N_j, is interrupted by an adjacent node that was timed out, N_i, it synchronizes with that node with a relative initial precision of $\Delta_{ij} = d$.*
Proof – By the Lemma *JoinTimedOut*, $\Delta_{ij} = d + \delta(D)$. Since $\rho = 0$ and $d \geq 0$, $\Delta_{ij} = d$. ■

Lemma *ρdJoinInterrupted – For $\rho = 0$, when a node, N_j, is interrupted by an adjacent node that was in turn interrupted, N_i, it synchronizes with that node with a relative initial precision of $\Delta_{ij} = \gamma$.*
Proof – By the Lemma *JoinInterrupted*, $\Delta_{ij} = \gamma + \delta(\gamma)$. Since $\rho = 0$ and $d \geq 0$, $\Delta_{ij} = \gamma$. ■

Lemma *ρdDeltaijAboutGamma – For $\rho = 0$, for all $t \geq 2P+\gamma$ clock ticks, the relative initial precision of any two adjacent nodes N_i and N_j is $\Delta_{ij} = \gamma$.*
Proof – By the Lemma *DeltaijAboutGamma*, $\Delta_{ij} = \gamma + \delta(\gamma)$. Since $\rho = 0$ and $d \geq 0$, $\Delta_{ij} = \gamma$. ■

Lemma *ρdInitialPrecision – For $\rho = 0$, for all $t \geq C_{Init}$, where $C_{Init} = 2P+K\gamma$ clock ticks, and upon the subsequent resynchronization process, the initial network precision is $\Delta_{Init} = (K\text{-}1)\gamma$.*
Proof – By the Lemma *InitialPrecision*, for all $t \geq C_{Init}$, $\Delta_{Init} = (K\text{-}1)(\gamma + \delta(\gamma))$. Since $\rho = 0$ and $d \geq 0$, $\Delta_{Init} = (K\text{-}1)\gamma$. ■

Lemma *ρdInitGuaranteedPrecision – For $\rho = 0$ and for all $t \geq C$, the initial guaranteed precision is $\Delta_{InitGuaranteed} = Wd$.*
Proof – By the Theorems *InitialConvergence* and *InitialClosure*, the network has converged, is synchronized, and remains synchronized for all $t \geq C_{Init}$ and upon subsequent resynchronization processes, the network converges to $\Delta_{Net}(t) = \Delta_{Init} + \delta(P)$. Since $\rho = 0$, $\Delta_{Net}(t) = \Delta_{Init}$. By the Lemma *ρdDeltaijAboutGamma* for all $t \geq C_{Init}$, all adjacent nodes are within $\Delta_{ij} = \gamma$ of each other. However, since $\rho = 0$ and $d \geq 0$, for all $t \geq C_{Init}$ and upon subsequent resynchronization processes, by the Lemma *ρdJoinTimedOut* at least two adjacent nodes will synchronize with $\Delta_{ij} = d$. Thus, at least one node per resynchronization

process, i.e., per P, will synchronize with its adjacent nodes with $\Delta_{ij} = d$. Repeating this process for $\lceil \Delta_{Init}/\gamma \rceil = W$ periods, will result in a synchronized network with a precision of $\Delta_{ij} = d$ for any two adjacent nodes N_i and N_j and an initial guaranteed precision of $\Delta_{InitGuaranteed} = Wd$ for the entire network. ■

Corollary *ρdPrecision – For $\rho = 0$, the precision of the network is $\pi = Wd$.*
Proof – By the Lemma *ρdInitGuaranteedPrecision*, for $\rho = 0$, the network converges to $\Delta_{InitGuaranteed} = Wd$. Recall that $\pi = \Delta_{InitGuaranteed} + \delta(P)$. Since $\rho = 0$ and $d \geq 0$, $\pi = Wd$. ■

Corollary *ρdConvergenceTime – For $\rho = 0$, the convergence time is $C = C_{Init} + \lceil \Delta_{Init} / \gamma \rceil P$.*
Proof – By the Lemma *ρdInitGuaranteedPrecision*, for $\rho = 0$, the network converges to $\Delta_{InitGuaranteed} = Wd$ within $C = C_{Init} + \lceil \Delta_{Init} / \gamma \rceil P$. ■

Note that when $d = 0$, i.e., for the ideal scenario, $\gamma = D$, $\Delta_{InitGuaranteed} = 0$, and $\pi = 0$. We now state the general lemmas and theorems that encompass all scenarios, i.e., for $0 \leq \rho << 1$ and $d \geq 0$ and for all $t \geq C$.

Theorem *Convergence – For all $t \geq C$, the network converges to a state where the guaranteed network precision is π, i.e., $\Delta_{Net}(t) \leq \pi$.*
Proof – By definition, for all $t \geq C$, $\pi = \Delta_{InitGuaranteed} + \delta(P)$. By the Theorem *InitialConvergence*, for all $t \geq C_{Init}$, $\pi = \Delta_{Init} + \delta(P)$. For $0 \leq \rho << 1$, it also follows that upon the completion of subsequent resynchronization processes, i.e., for all $t \geq C$ where $C \geq C_{Init}$, the initial guaranteed precision of the network is Δ_{Init} and upon subsequent resynchronization processes, no further improvement on Δ_{Init} can be guaranteed. Therefore, no improvement on $\Delta_{InitGuaranteed}$ can be achieved. Also, by the Corollary *InitGuaranteedPrecision*, $\Delta_{InitGuaranteed} = \Delta_{Init}$ and so no improvement on π can be guaranteed. Also, since the maximum drift for a time interval of P is bounded by $\delta(P)$, the network has converged with precision π and remains within π. ■

Theorem *Closure – For all $t \geq C$, a synchronized network where all nodes have converged to $\Delta_{Net}(t) \leq \pi$, shall remain within the synchronization precision π.*
Proof – By the Theorem *Convergence*, for all $t \geq C$, the network converges with the precision $\pi = \Delta_{InitGuaranteed} + \delta(P)$. It also follows that for $0 \leq \rho << 1$ and for all $t \geq C$, upon subsequent resynchronization processes the network precision remains within π. Therefore, the network has converged, is synchronized, and remains synchronized within π. ■

Theorem *Congruence – For all nodes N_i and for all $t \geq C$, (N_i.LocalTimer(t) = γ) implies $\Delta_{Net}(t) \leq \pi$.*
Proof – By the protocol (part *E3*) upon time out a node resets its *LocalTimer* to zero. By the Lemma *IncLocalTimer* the *LocalTimer* of a node will always get incremented and, unless interrupted, a timed out node will reach γ. By the protocol (parts *E1*, *E2*) when a node gets interrupted, it resets its *LocalTimer* to γ. Thus, a node will always reach γ. By the Theorems *Convergence* and *Closure*, for all $t \geq C$, the network convergences and remains synchronized with $\pi = \Delta_{InitGuaranteed} + \delta(P)$. Thus, for all $t \geq C$ and $0 \leq \rho << 1$, when N_i.*LocalTimer(t)* = γ, the network precision remains within π, i.e., $\Delta_{Net}(t) \leq \pi$, and all nodes are in synchrony. ■

Lemma *InitGuaranteedPrecision – For all $t \geq C$, the initial guaranteed precision of the network is $Wd \leq \Delta_{InitGuaranteed} \leq (K\text{-}1)(\gamma+\delta(\gamma))$, where $\Delta_{InitGuaranteed} = Wd$, for $\rho = 0$, and $\Delta_{InitGuaranteed} \leq (K - 1)(\gamma+\delta(\gamma))$, for $\rho > 0$.*
Proof – By the Lemma *InitialPrecision*, for $0 \leq \rho << 1$ and for all $t \geq C_{Init}$ and upon the subsequent resynchronization process, the initial network precision is $\Delta_{Init} = (K\text{-}1)(\gamma + \delta(\gamma))$. By the Theorem *InitialConvergence*, for $0 \leq \rho << 1$ and $t \geq C_{Init}$ and upon the completion of the subsequent resynchronization processes, the initial guaranteed precision of the network is Δ_{Init}, in other words, $\Delta_{InitGuaranteed} = \Delta_{Init}$. By the Lemma *ρdInitGuaranteedPrecision*, for $\rho = 0$ and for all $t \geq C$, the initial guaranteed precision is $\Delta_{InitGuaranteed} = Wd$. Thus, for $0 \leq \rho << 1$ and for all $t \geq C$, the initial guaranteed precision is bounded by $Wd \leq \Delta_{InitGuaranteed} \leq (K - 1)(\gamma+\delta(\gamma))$. ■

Theorem *GuaranteedPrecision – For for all $t \geq C$, the guaranteed precision of the network is bounded by $Wd \leq \pi \leq \Delta_{InitGuaranteed} + \delta(P)$.*
Proof – By the Lemma *InitGuaranteedPrecision*, for all $t \geq C$ and $0 \leq \rho << 1$, the initial network precision is bounded by $Wd \leq \Delta_{InitGuaranteed} \leq (K - 1)(\gamma+\delta(\gamma))$ with the lower bound for $\rho = 0$ and the upper bound for $\rho > 0$. By definition, for all $t \geq C$, $\pi = \Delta_{InitGuaranteed} + \delta(P)$. Therefore, for $0 \leq \rho << 1$ and for all $t \geq C$, the guaranteed precision of the network is bounded by $Wd \leq \pi \leq \Delta_{InitGuaranteed} + \delta(P)$. ■

Note that when $d = 0$, i.e., for the ideal scenario, $\gamma = D$, $\Delta_{InitGuaranteed} = 0$, and $\pi = 0$.

Lemma *ConvergenceTime – The convergence time is $C = C_{Init} + \lceil \Delta_{Init}/\gamma \rceil P$.*
Proof – By the Theorem *InitialConvergence*, for $0 \leq \rho << 1$, the network converges within C_{Init}. By the Corollary *ρdConvergenceTime*, for $\rho = 0$, the convergence time is $C = C_{Init} + \lceil \Delta_{Init}/\gamma \rceil P$. Since the convergence time is an upper bound, for $0 \leq \rho << 1$, the convergence time is $C = C_{Init} + \lceil \Delta_{Init}/\gamma \rceil P$. ■

Theorem *Liveness – For all $t \geq C$, LocalTimer of every node sequentially takes on at least all integer values in $[\gamma, P - \pi]$.*
Proof – By the Theorems *Convergence* and *Closure*, for all $t \geq C$, a synchronized network where all nodes have converged to π, remains within π. Since the network is synchronized, all nodes either time out or get interrupted by a timed out adjacent node within π. For the worst case analysis, when the fastest node reaches P, times out and transmits a new *Sync* message, the slowest node is at $P - \pi$. If the slowest node is adjacent to the fastest node, it gets interrupted within the next γ. If the slowest node is farthest away from the fastest node, it times out within π. If the slowest node gets interrupted before reaching P, by the protocol (part *E1*, *E2*), it sets its *LocalTimer* while accounting for the message arrival delay, i.e., *LocalTimer* $= \gamma$. For the worst case analysis of the liveness property, the slowest node is adjacent to the fastest node. Since the network is in synchrony, the slowest node will not get interrupted again until a fastest node times out within the next P. Therefore, the slowest node sequentially takes on all integer values in $[\gamma, P - \pi]$. The fastest node, however, sequentially takes on all integer values in $[0, P]$. Furthermore, upon convergence this process repeats during every P time interval and after every resynchronization process. Thus, for all $t \geq C$, all nodes sequentially take on at least all integer values in $[\gamma, P - \pi]$. ■

5. Conclusions

In this report, we presented a deductive proof of the correctness of a self-stabilizing distributed clock synchronization protocol that is focused on the distributed synchronization of an arbitrary, fault-free, and non-partitioned digraph ranging from fully connected to 1-connected networks of nodes while allowing for differences in the network elements. We presented a deductive proof of the correctness of the protocol as it applies to the networks with unidirectional and bidirectional links. We also confirmed the claims of determinism and linear convergence. This protocol does not rely on assumptions about the initial state of the system and no central clock or centrally generated signal, pulse, or message is used. Nodes are anonymous, i.e., they do not have unique identities. There is no theoretical limit on the maximum number of participating nodes. The only constraint on the behavior of the node is that the interactions with other nodes are restricted to defined links and interfaces.

We have shown and proven how to synchronize an arbitrary digraph in the absence of faults. This effort brought up the following questions. Can an arbitrary digraph be synchronized in the presence of faults? What types of faults can an arbitrary digraph tolerate? Looking at the problem from a different perspective, if the faults are symmetric, what types of graphs can synchronize in their presence? What if the faults are asymmetric (Byzantine), what types of graphs can synchronize in their presence?

References

[Bar 2002] Barahona, M.; Pecora, L.M.: “Synchronization in Small-World Systems,” Phys. Rev. Lett. 89 (2002) 054101, 2002.

[But 2008] Butler, R.: “A primer on architectural level fault tolerance,” NASA/TM-2008-215108, February 2008.

[Gad 2000] Gade, P.M.; Hu, C.K.: “Synchronous chaos in coupled map lattices with small-world interactions,” Phys. Rev. E 62 (2000) 6409–6413, 2000.

[Gir 2005] Girault, A.; Rutten, E.: “Modeling Fault-tolerant Distributed Systems for Discrete Controller Synthesis,” Electronic Notes in Theoretical Computer Science, vol. 133, pp. 81-100, 2005.

[Gom 2007] Gomez-Gardenes, J.; Moreno, Y.; Arenas, A.: “Paths to Synchronization on Complex Networks,” Phys. Rev. Lett. 98 (2007), 034101, 2007.

[Hon 2002] Hong, H.; Choi, M.Y.; Kim, B.J.: “Synchronization on small-world networks,” Phys. Rev. E 65 (2002) 026139, 2002.

[Hon 2004] Hong, H.; Kim, B.J.; Choi, M.Y.; Park, H.: “Factors that predict better synchronizability on complex networks,” Phys. Rev. E 69 (2004) 067105, 2004.

[Kop 1997] Kopetz, H: “Real-Time Systems, Design Principles for Distributed Embedded Applications,” Kluwar Academic Publishers, ISBN 0-7923-9894-7, 1997.

[Li 2004] Li, C.; Chen, G.: “Phase synchronization in small-world networks of chaotic oscillators,” Physica A 341 (2004) 73–79, 2004.

[Mal 2006A] Malekpour, M.R.: A Byzantine-Fault Tolerant Self-Stabilizing Protocol for Distributed Clock Synchronization Systems. Eighth International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS06), November 2006.

[Mal 2006B] Malekpour, M.R.: “A Byzantine-Fault Tolerant Self-Stabilizing Protocol for Distributed Clock Synchronization Systems.” NASA/TM-2006-214322, pp. 37, August 2006.

[Mal 2011A] Malekpour, M.R.: “A Self-Stabilizing Distributed Clock Synchronization Protocol For Arbitrary Digraphs”. NASA/TM-2011-217054, pp. 42, February 2011.

[Mal 2011B] Malekpour, M.R.: “Model Checking A Self-Stabilizing Distributed Clock Synchronization Protocol For Arbitrary Digraphs”. NASA/TM-2011-217152, pp. 31, May 2011.

[Tor 2005] Torres-Pomales, W; Malekpour, M.R.; Miner, P.S.: ROBUS-2: A fault-tolerant broadcast communication system. NASA/TM-2005-213540, pp. 201, March 2005.

[Wat 1998] Watts, D.J.; Strogatz, S.H.: “Collective dynamics of ’small-world’ networks,” Nature (London) 393, 440, 1998.

Appendix A. Symbols

The symbols used in the protocol are described in detail in [Malekpour 2010] and are listed here for reference.

Symbols	Descriptions
K	sum of all nodes
T	network topology
D	event-response delay
d	network imprecision
ρ	bounded drift rate with respect to real time
P	self-stabilization/synchronization period
F	sum of all faulty nodes
N_i	the ith node
M_i	the ith monitor of a node
γ	communication latency
L	the largest loop in the graph
W	the width or diameter of the graph
T_S	graph threshold
π	the guaranteed self-stabilization/synchronization precision
C	convergence time
C_{Init}	time of initial synchrony
LocalTimer	node's local logical clock
Δ_{ij}	precision between *LocalTimers* of any two adjacent nodes N_i and N_j
Δ_{Init}	initial precision among *LocalTimers* of all nodes immediately after a resynchronization process
$\Delta_{InitGuaranteed}$	initial guaranteed precision among *LocalTimers* of all nodes immediately after a resynchronization process
$\delta(t)$	drift per t
Sync	self-stabilization/synchronization message
$\Delta_{Net}(t)$	precision among *LocalTimers* of all nodes at time t

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)* 01- 10 - 2011	**2. REPORT TYPE** Technical Memorandum	**3. DATES COVERED** *(From - To)*

4. TITLE AND SUBTITLE Correctness Proof of a Self-Stabilizing Distributed Clock Synchronization Protocol for Arbitrary Digraphs	**5a. CONTRACT NUMBER**
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Malekpour, Mahyar R.	**5d. PROJECT NUMBER**
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER 534723.02.02.07.30
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) NASA Langley Research Center Hampton, VA 23681-2199	**8. PERFORMING ORGANIZATION REPORT NUMBER** L-20066
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) National Aeronautics and Space Administration Washington, DC 20546-0001	**10. SPONSOR/MONITOR'S ACRONYM(S)** NASA
	11. SPONSOR/MONITOR'S REPORT NUMBER(S) NASA/TM-2011-217184

12. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified Unlimited
Subject Category 62
Availability: NASA CASI (443) 757-5802

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This report presents a deductive proof of a self-stabilizing distributed clock synchronization protocol. It is focused on the distributed clock synchronization of an arbitrary, non-partitioned digraph ranging from fully connected to 1-connected networks of nodes while allowing for differences in the network elements. This protocol does not rely on assumptions about the initial state of the system, and no central clock or a centrally generated signal, pulse, or message is used. Nodes are anonymous, i.e., they do not have unique identities. There is no theoretical limit on the maximum number of participating nodes. The only constraint on the behavior of the node is that the interactions with other nodes are restricted to defined links and interfaces. We present a deductive proof of the correctness of the protocol as it applies to the networks with unidirectional and bidirectional links. We also confirm the claims of determinism and linear convergence.

15. SUBJECT TERMS

Algorithm; Clock synchronization; Communication network; Digraph; Distributed; Emergent system; Protocol; Self-stabilizing

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			STI Help Desk (email: help@sti.nasa.gov)
					19b. TELEPHONE NUMBER *(Include area code)*
U	U	U	UU	31	(443) 757-5802