NASA/TM-2011-217054

A Self-Stabilizing Distributed Clock Synchronization Protocol for Arbitrary Digraphs

Mahyar R. Malekpour
Langley Research Center, Hampton, Virginia

February 2011

NASA STI Program . . . in Profile

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*

- 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

NASA/TM-2011-217054

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

February 2011

Acknowledgments

This effort was conducted under the Integrated Vehicle Health Management (IVHM) project of NASA‟s Aviation Safety program and was made possible by the support from Eric G. Cooper, Associate Principal Investigator for NASA‟s IVHM Project. The author would like to thank the reviewers for their helpful comments. The author would like to especially thank Cesar Munoz for his in-depth review and constructive comments.

Available from:

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

Abstract

This report presents a self-stabilizing distributed clock synchronization protocol in the absence of faults in the system. 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, other than the presence of at least one node, 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 an outline of a deductive proof of the correctness of the protocol. A model of the protocol was mechanically verified using the Symbolic Model Verifier (SMV) for a variety of topologies. Results of the mechanical proof of the correctness of the protocol are provided. The model checking results have verified the correctness of the protocol as they apply to the networks with unidirectional and bidirectional links. In addition, the results confirm the claims of determinism and linear convergence. As a result, we conjecture that the protocol solves the general case of this problem. We also present several variations of the protocol and discuss that this synchronization protocol is indeed an emergent system.

Table of Contents

1. Historic Perspective

How can a distributed system solve a problem that is inherently global by executing a set of rules locally? For millennia people have witnessed in awe flocks of birds fly in unison, hundreds of frogs croak in harmony and thousands of fire flies flash in synchrony and wondered how such collective (or mass) synchrony comes about. In Southeast Asia, a large number of *Malaccae* fireflies routinely flash on and off in synchrony. Do these insects follow a leader or do they have an inherit sense of rhythm? This question was asked by George Hudson in 1918 [Hud 1918] [Str 2003]. The synchronization phenomenon, whether a natural occurrence or artificially induced, still fascinates us today and has become one of the most interesting scientific problems of our time. In a recent survey, Arenas [Are 2008] reports on the advances in the understanding of synchronization phenomena by oscillating elements in a complex network topology. He concludes that: "Synchronization processes are ubiquitous in nature and play a very important role in many different contexts as biology, ecology, climatology, sociology, technology, or even in arts." But, how does collective synchrony emerge from chaos? The answer to this question has intrigued mankind, in particular, some of the greatest minds of the twentieth century, including Albert Einstein, Richard Feynman, Norbert Wiener, Brian Josephson, Edward Lorenz, and Arthur Winfree. Many questions still persist today. Is synchrony inevitable? If so, how exactly does it happen? When and under what circumstances is it possible or impossible to achieve? What are the ramifications of either case? What are the theoretical and practical implications of either case?

Besides being an intellectual curiosity and a theoretical problem in computer science and engineering, synchronization has practical significance as a fundamental service for higher-level algorithms that solve other problems. For example, in safety-critical TDMA (Time Division Multiple Access) architectures [Kop 1997] [Min 2002] [Tor 2005A, 2005B], synchronization is the most crucial element of these systems.

Clock synchronization algorithms are essential for managing the use of resources and controlling communication in a distributed system. We define **synchronization** of a distributed system as the process of **achieving** and **maintaining** coordination among independent local clocks by exchanging local time information. We define **bounded-synchrony** as the exchange of local time information by the nodes in unison but within a given bound. True synchrony, as operating and exchanging messages in perfect unison, is only possible under strictest assumptions and ideal conditions. Bounded-synchrony on the other hand is a more general term that encompasses imperfections in the network. Hereafter in this report, we use the term **synchrony** to mean bounded-synchrony.

Charlie Peskin [Pes 1975] posed the self-organization idea around 1975 while working on cardiac pacemakers and, at about the same time, Edsger Dijkstra [Dij 1974] presented the self-stabilization problem in a distributed system. These two scientists asked whether it would be possible for a set of oscillators or machines to self-organize and self-stabilize their collective behavior in spite of unknown initial conditions and distributed control.

A distributed system is defined 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 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. Hereafter in this report, we use the term synchronization to mean self-stabilizing clock synchronization in distributed systems.

There is a vast literature on synchronization phenomenon exhibited by humans, animals, and even inanimate objects. There are also many proposed solutions for synchronization of a large number of entities based on models inspired by nature or abstract ideas. In [Are 2008] Arenas *et al.* reports on the advances in the comprehension of synchronization phenomena in the context of a complex network topology and presents extensive numerical work as well as analytical approaches to the synchronization problem and reviews several applications of synchronization to complex networks in various disciplines from biological systems to social sciences.

There exist many solutions for special cases and restricted conditions. For example, Strogatz *et al.* provide a solution when the oscillators are nearly identical, perturbations are absent, and all oscillators are coupled equally to one another [Mir 1990] [Str 2003]. In other words, the network is a fully connected graph under ideal conditions. Such restrictions were necessary to make the dynamics of the system mathematically tractable. In [Nis 2006A, 2006B] the solution assumes an unidirectional information flow so that the networks become optimally synchronizable. Other solutions require embedding a directed spanning tree or rewiring the network in order to achieve synchrony [Gle 2006] [Nis 2006A, 2006B] [Bre 2008]. In [Ear 2003], a solution for the general case is presented, but a closer examination reveals that it only addresses maintaining synchronization (stability of stable in-phase synchronization) and not how to achieve it. Furthermore, although their solution applies to a random graph, they require each node to be connected to four other nodes. In computer science and computer engineering terminology, stability is referred to as the closure property. The **convergence** and **closure** properties address *achieving* and *maintaining* network synchrony, respectively (see Section 4.3 for a formal definition of these parameters). There are many solutions that deal with the closure property [Lam 1985] [Sri 1987] [Wel 1988] and either do not address convergence or provide an ad hoc solution [Dav 1978] for initialization and integration, separately. Typically, the assumed topology is a regular[1] graph such as a fully connected graph or a ring. These topologies do not necessarily correspond to practical applications or biological, social, or technical networks. Furthermore, the existing models and solutions do not always achieve synchrony and, therefore, do not solve the general case of the distributed synchronization problem. Even when the solutions achieve synchrony, the time to achieve synchrony is very large for many of the solutions.

Another key factor in a proposed solution is whether or not it deals with faults. A **fault** is a defect or flaw in a system component resulting in an incorrect state [Gir 2005] [Tor 2005A] [But

[1] A regular graph is a graph where each vertex has the same number of neighbors, i.e., every vertex has the same degree or valency. A regular graph with vertices of degree *k* is called a *k*-regular graph or regular graph of degree *k*.

2008]. Large-scale distributed systems have become an integral part of safety-critical computing applications, necessitating system designs that incorporate complex fault-tolerant resource management functions to provide globally coordinated operations with ultra-reliability. As a result, robust clock synchronization has become a required fundamental component of fault-tolerant safety-critical distributed systems. The requirement to handle faults adds a new dimension to the complexity of the synchronization of fault-tolerant distributed networks. Ultra-reliable distributed systems are designed to deal with variety of faults that reflect the desired degree of reliability of the system. Although the solutions for other systems consider perturbations, they do not necessarily address faulty behaviors in the network. We define the **fault spectrum** as a range of faults that span from no faulty nodes at one extreme end to arbitrary (Byzantine) faulty nodes at the other extreme end.

A fundamental property of a robust distributed system is the capability of tolerating and potentially recovering from failures that are not predictable in advance. In [Lam 1982, 1985] various ideas for overcoming failures in a robust distributed system are addressed that include tolerating Byzantine faults. There are many algorithms that address permanent faults [Sri 1987], where the issue of transient failures is either ignored or inadequately addressed. There are many efficient Byzantine clock synchronization algorithms proposed that are based on assumptions on initial synchrony of the nodes [Sri 1987] and [Wel 1988] or existence of a common pulse at the nodes, e.g., the first protocol in [Dol 2004]. There are many clock synchronization algorithms that are based on randomization and, therefore, are non-deterministic, e.g., the second protocol in [Dol 2004]. In [Mal 2006A] a counterexample is presented to a clock synchronization algorithm [Dal 2003] that is based on the existence of a common pulse at the nodes.

A Byzantine-Fault-Tolerant Self-Stabilizing Protocol for Distributed Clock Synchronization Systems was reported in [Mal 2006B]. Claims about the protocol were validated via mechanical verification of a system consisting of one permanent Byzantine faulty node [Mal 2008]. This protocol synchronizes a fully connected network of two or more nodes in the absence of faults. A Self-Stabilizing Byzantine-Fault-Tolerant Clock Synchronization Protocol was reported in [Mal 2009]. This protocol also synchronizes a fully connected network of two or more nodes in the absence of faults. Instances of these protocols are demonstrated to self-stabilize from any state, in the presence of at most one permanent Byzantine faulty node, and deterministically converge in linear time with respect to the synchronization period. These protocols, however, do not solve the general case of the problem in the presence of multiple Byzantine faults.

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 the solution. 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. 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. Furthermore, addressing network element imperfections is necessary to make a solution applicable to realizable systems.

In this report, we present a solution for an arbitrary, non-partitioned network (digraph) in the absence of faults. The networks range 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) formation. We do not require any particular information flow nor imposes changes to the network in order to achieve synchrony. However, we focus on one extreme of the fault spectrum and only consider distributed systems in the absence of faults. The assumption of an absence of faults is equivalent to the assumption that all faults are detectable. This departure from the Byzantine extreme of the fault spectrum 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. To summarize, the rationale for this approach was 1) to reduce the problem to a more manageable size, 2) to search for a general solution in the absence of faults before attempting to solve the problem in the presence of various faults, 3) and to solve the problem that is applicable to a majority of applications.

In section 2 of this report, we provide a system overview. We present the protocol description in section 3 and present the protocol in section 4. In section 5 we present results of mechanical proof of the protocol via model checking. In section 6 we discuss variations of the protocol including the general case of the protocol that encompasses dynamic node count and dynamic topology. We also discuss the bounds on the drift rate of the oscillators. Finally, we present concluding remarks in section 7 and enumerate possible applications.

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 channels that represent their interconnectivity.

The underlying topology considered here is a network of $K \geq 1$ nodes that exchange messages through a set of communication channels that represent their interconnectivity. 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 channels are assumed to connect a set of source nodes to a set of destination nodes with a source node being different than a destination node. All communication channels 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 to other nodes is realized by transmitting the message to all connected nodes at the same time. 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 channels and nodes can behave arbitrarily provided that eventually the system adheres to the protocol assumptions (see Section 4.2).

2.1. Drift Rate (ρ) And The Logical Clock (*LocalTimer*)

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 2005A].

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 expressed as $0 \leq \rho << 1$. 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.

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

2.2. Communication Delay (*D*), Network Imprecision (*d*), And γ

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 in this figure, a message transmitted at real time t_0 is expected to arrive at all destination nodes, be processed, and subsequent messages are 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 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 discrete values with units of real time clock tick. The communication latency, denoted γ, is expressed in terms of *D* and *d*, and is constrained by $\gamma = (D+d)$.

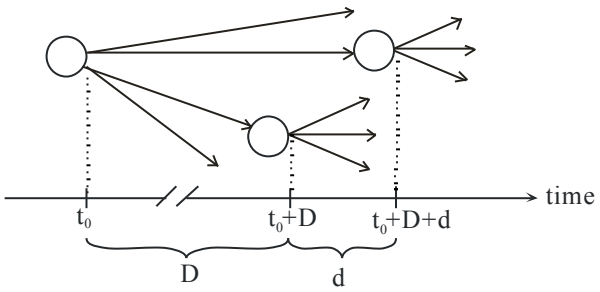

Figure 1. Event-response delay, *D*, and network imprecision, *d*.

2.3. Topology (*T*)

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

In this report, we use the terms network and graph interchangeably as well as the terms link, channel and edge. The number of strongly connected directed graphs for a given set of nodes have been studied by Liskovets [Lis 1970]. Since the number of digraphs is exceedingly large for small values of *K*, for model checking purposes we‟ve also considered subsets of the digraphs. In particular, the set of graphs with only bidirectional links is a subset of all digraphs. An even smaller set is the set of *x*-connected graphs where $x \geq 1$. Of particular interest is the set of 1-connected graphs where each node is connected to the graph by at least one bidirectional channel. Table 1 provides a count of possible graphs with bidirectional links for $K = 1$ through 19 nodes [Sloane] and the corresponding count of digraphs. As is evident from the table, the number of possible 1-connected graphs, *a(K)*, grows exponentially as *K* increases linearly. The number of digraphs grows at even much faster rate compared to *a(K)*.

Table 1. Number of graphs for a given *K*.

	Number Of Graphs With Only Bidirectional Links (Sloane's A001349)	**Number Of Digraphs**
K	***a(K)***	**-**
1	1	1
2	1	1
3	2	5
4	6	83
5	21	5048
6	112	1047242
7	853	
8	11117	
9	261080	
10	11716571	
11	1006700565	
12	164059830476	
13	50335907869219	
14	29003487462848061	
15	31397381142761241960	
16	6396956011322517617627 7	
17	24587183168208402651952 8	
18	17873317252488990888902 00576580	
19	24636021429399867655322650759681644	

The following graph specific terms are used in the subsequent sections of this report.

- 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* denotes 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* signifies the **width** or diameter of the graph, i.e., the maximum value of the shortest path connecting any two nodes.

Table 2 provides a list of selected graphs with bidirectional links and their corresponding L and W values. In general, for digraphs, L and W are at their maximum, i.e., $L = K$ and $W = K - 1$.

Table 2. L and W for graphs with only bidirectional links.

Graph (bidirectional links)	***L***	***W***
A single node	1	0
Linear	2	$K - 1$
Star	2	2
Ring, singly connected	K	$\lceil K / 2 \rceil$
Ring, doubly connected	K	$\lceil K / 4 \rceil$
Grid ($a \times b$, $a \leq b$)	K	$a + b - 2$
Full Grid ($a \times b$, $a \leq b$)	K	$b - 1$
Fully Connected	K	1
Random	$1 \leq L \leq K$	$0 \leq W \leq K - 1$
Bipartite	$K - 1$	2

3. Protocol Description

In this section we provide a description of the protocol and provide an intuitive depiction of its behavior. The system has two synchronization states: **synchronized** and **unsynchronized**. The system is in the unsynchronized state when it starts up, i.e., at power-on. The system is in the synchronized state when the nodes are within an expected bounded precision. The system transitions from the unsynchronized state to the synchronized state after the execution of a synchronization protocol. Therefore, the clock synchronization protocol is expected to enable the system to transition to the synchronized state and remain there. When a system reaches and operates in the synchronized state, it is said to be synchronous or in synchrony. Due to the inherent drift in the local times, a synchronization protocol must be re-executed at regular intervals to ensure that the local times are kept synchronized. The rate of resynchronization is constrained by physical parameters of the design (e.g., oscillator drift rates) as well as precision and accuracy goals. The protocol presented in this report addresses achieving and maintaining the precision goal of the system. Achieving the clock *accuracy* goal is beyond the scope of this report and is addressed separately as described in [Mal 2006B]. Therefore, the clock synchronization protocol enables the system to achieve and maintain synchrony among distributed local logical clocks, i.e., *LocalTimers* (not local physical oscillators).

The following definition for resynchronization process is from [Mal 2009] and is rephrased here for reference. 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. The periodic synchronization after achieving the initial synchrony is referred to as the **resynchronization process** whereby all nodes reengage in the synchronization process. A node is said to **time-out** when its *LocalTimer* reaches its maximum value. The resynchronization process begins when the first node (fastest node) times-out and transmits a *Sync* message and ends after the last node (slowest node) transmits a *Sync* message. For $\rho << 1$, the fastest node cannot time-out again before the slowest node transmits a *Sync* message (see Section 6.2 for more discussion on ρ). A node is said to be **interrupted** when it accepts an incoming *Sync* message before its *LocalTimer* reaches its maximum value, i.e., before it times-out.

A node consists of a **synchronizer** and a set of **monitors**. 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 channels.

The following definitions and terms are used in the description and operation of the protocol presented in this report. All protocol parameters have discrete values with the time-based terms having units of real time clock ticks. The discretization is for practical purposes in implementing and model checking of the protocol. Although, the network level measurements are real values, locally and at the node level, all parameters are discrete.

- 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 at time t is denoted by $\Delta_{ij}(t)$ 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 at time t is by denoted $\Delta_{Init}(t)$, has units of real time clock ticks and is defined as a measure of the precision of the network after elapse time of C_{Init}.
- The **initial guaranteed precision** among *LocalTimers* of all nodes at time t is denoted by $\Delta_{InitGuaranteed}(t)$, has units of real time clock ticks and is a measure of the precision of the network after elapse time of C.
- The maximum number of faulty nodes is denoted as F.

3.1. How Does The Protocol Work?

In this section we provide an intuitive description of the protocol behavior. A node periodically undergoes a resynchronization process either when its *LocalTimer* times out or when it receives a *Sync* message. If it times out, it broadcasts a *Sync* message and so initiates a new round of a resynchronization process. However, since we are assuming that there are no faults present, i.e., $F = 0$, when a node receives a *Sync* message, except during a predefined window, it accepts the *Sync* message and undergoes the resynchronization process where it resets its *LocalTimer* and relays the *Sync* message to others. This process continues until all nodes participate in the resynchronization process and converge to a guaranteed precision. The predefined window where the node ignores all incoming *Sync* messages, referred to as **ignore window**, provides a means for the protocol to stop the vicious cycle of resynchronization processes triggered by the follow up *Sync* messages.

To provide an insight into the behavior of network we draw analogy from a pool of water. A pool of undisturbed water remains calm. Dropping a rock in a tranquil pool of water generates a wave which ripples toward the outer edges of the pool with the center of the wave at the point where the rock disturbed the still water. Assuming no other disturbance sources (including the edges of the pool), eventually, the ripple fades away and the pool returns to tranquility. We assume a node in a distributed network emanates a flashing light when it transmits a *Sync* message. In the absence of drift, when the network is in synchrony, it remains in perfect synchrony. As the nodes go through the resynchronization processes, they pulsate flashes of light at regular intervals and in perfect unison. However, in the presence of drift, as the nodes go through the resynchronization processes, the fastest node transmits a *Sync* message and the associated flash of light before the other nodes. In this scenario, the fastest node disturbs the pool of tranquility by generating a new wave that ripples through the system. We refer to this phenomenon as the **ripple effect**. The ripple effect is more pronounced as the relative drift of the nodes increases. The network returns to the tranquility state when the ripple effect wears out. Note that there may be multiple ripples in the system originating from as many nodes.

3.2. The Graph Threshold (T_S)

The graph threshold, T_S, is a function of a specified graph topology, i.e., $T_S = f(T)$ and is given by the following equation.

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

Defining T_S in terms of L requires knowledge of the topology of the given network. From Table 2, $L \leq K$, i.e., in the worst case, $L = K$. Thus, 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 K. However, for a specific application, optimizing T_S by expressing it in terms of L results in faster synchrony and better performance.

3.3. Sync Message

In order to achieve synchrony, the nodes communicate by exchanging *Sync* messages. Since only one message type is used for the operation of this protocol, a single bit suffices. When the system is in synchrony, the protocol overhead is at most one message per resynchronization period P.

3.4. Message Validity

Only one message type is required for the operation of the protocol. Assuming physical-layer error detections are dealt with separately, receiving 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 [Mal 2009]. A valid

Sync message is discarded after it is relayed to the synchronizer and has been kept for one local clock tick.

3.5. The Monitor

To assess the behavior of other nodes, a node employs as many monitors as the number of nodes it is connected to with one monitor for each source of incoming message. Figure 2 depicts a scenario for a fully connected graph where a node has (K-1) monitors. 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, time-stamps, 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 (Sections 3.6 and 4).

3.6. The Synchronizer

The assessment results of the monitored nodes are utilized by the node in the synchronization process. The synchronizer describes the behavior of the node, N_i, utilizing assessment results from its monitors, as shown in Figure 2, where Monitor_j, $i \neq j$, is the monitor for the corresponding node N_j.

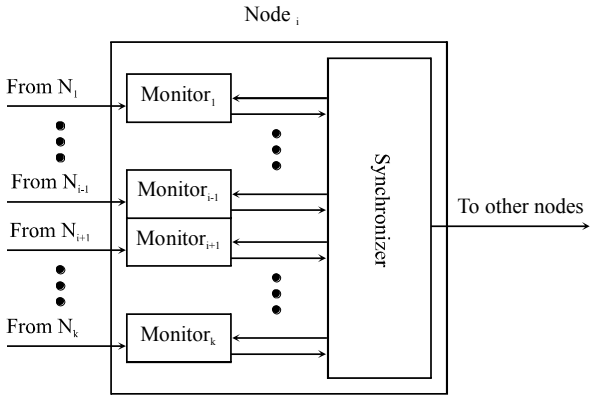

Figure 2. The i[th] node, N_i, with its monitors and synchronizer.

4. The Protocol

In this section we enumerate protocol assumptions, properties, parameters, and describe the protocol in pseudo-code. As we have elaborated thus far in previous sections, the general form of the distributed synchronization problem, *S*, is defined by the following septuple.

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

In other words, the distributed synchronization problem is a function of the number of nodes, network topology, communication delay, communication imprecision, oscillator drift rate, synchronization period, and number of faults, respectively. The solution to this problem is a protocol with convergence and closure properties, at a minimum, as discussed subsequently in this section.

4.1. Protocol Function

The functions used in the protocol are described in this section. The function *ValidateMessage()* 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.

4.2. Protocol Assumptions

1. All nodes correctly execute the protocol.
2. All channels correctly transmit data from their sources to their destinations.
3. $K \geq 1$.
4. T = strongly connected digraph.
5. A message sent by a node will be received and processed by all other nodes within γ, where $\gamma = (D + d)$.
6. $0 \leq \rho << 1$.
7. Absence of faults in the links and nodes, i.e., $F = 0$.
8. 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.)

4.3. 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 $t_0 = 0$ 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,
- $\Delta_{Net}(t)$, for real time t, is the maximum difference of values of the *LocalTimers* of any two nodes (i.e., the relative clock skew) for $t \geq t_0$, and
- π, the synchronization precision, is the guaranteed upper bound on $\Delta_{Net}(t)$, for all $t \geq C$.

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.

$$\Delta_{Net}(t) = min\ ((LocalTimer_{max}(t) - LocalTimer_{min}(t)),\ (LocalTimer_{max}(t - r) - LocalTimer_{min}(t - r)))$$

where,

$r = (W + 1)\gamma$,

$LocalTimer_{min}(x) = min\ (N_i.LocalTimer(x))$, and

$LocalTimer_{max}(x) = max\ (N_i.LocalTimer(x))$, for all i.

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 $\Delta_{Net}(t) \leq \pi$).

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

The protocol is presented in Figure 3 and consists of a synchronizer and a set of monitors which execute once every local clock tick. The protocol is based on the fact that the network imprecision d is not restricted by an upper bound. If, however, we do restrict d by $0 \leq d \leq D$, then in statement *E1* the expression D can be replaced by γ.

Monitor:
case (message from the corresponding node)
{*Sync*:
    ValidateMessage()

Other:
    Do nothing.
} // case
ConsumeMessage()

Synchronizer:
E1: if (*ValidSync()* and (*LocalTimer* < D)
    LocalTimer := γ,

E2: elseif ((*ValidSync()* and (*LocalTimer* ≥ T_S))
    LocalTimer := γ,
    Transmit *Sync*,

E3: elseif (*LocalTimer* ≥ P) // *time-out*
    LocalTimer := 0,
    Transmit *Sync*,

E4: else
    LocalTimer := *LocalTimer* + 1.

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

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 $\rho = 0$

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

$P \geq \max((2K + 1)\gamma + \delta((2K + 1)\gamma), 3T_S + \delta(3T_S))$, 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.

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

$P \geq KT_S + \delta(KT_S)$

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

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

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

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

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

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

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

5. Proof Of The Protocol

There are two general formal methods approaches for the verification of the correctness of a protocol; **theorem proving** and **model checking**. Proof via theorem proving requires a deductive proof of the protocol. Proof via model checking is based on specific scenarios and generally limited to a subset of the problem space. A deductive proof of the protocol is the subject of a subsequent report. In the mean time, we chose the model checking approach for its ease, feasibility, and quick examination of a subset of the problem space while attempting a more comprehensive proof via theorem proving. Details of the model checking efforts will be the subject of a subsequent report.

What follows in this section is the model checking results of the proof of correctness of the protocol. In particular, model checking effort encompasses the verification of correctness of a model of the protocol by confirming that a candidate system self-stabilizes from any state. This effort, furthermore, includes the verification of claims of determinism and linear convergence of the model of the protocol with respect to the synchronization period.

The proof idea is depicted in Figure 4. The main theorems are enumerated here and 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* 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?

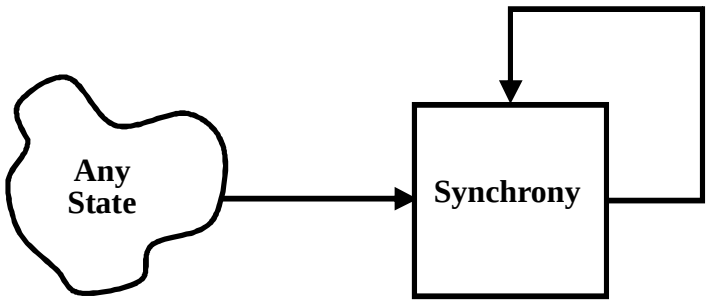

Figure 4. Proof approach.

Theorem *Convergence – After elapse time of C, the network converges to a state where the guaranteed network precision is π, i.e.,* $\Delta_{Net}(t) \leq \pi$.

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.*

Theorem *Congruence – For all nodes* N_i *and for all* $t \geq C$, (N_i.*LocalTimer*$(t) = \gamma$ *implies* $\Delta_{Net}(t) \leq \pi$).

Lemma *InitialPrecision – For* $\rho \geq 0$, *the initial precision of the network and after elapse time of* C_{Init} *is* $\Delta_{Init}(C_{Init}) \leq (K - 1)(\gamma+\delta(\gamma))$.

Lemma *InitGuaranteed – For* $\rho \geq 0$*, the initial guaranteed precision of the network and after elapse time of C is* $Wd \leq \Delta_{InitGuaranteed}(C) \leq W(\gamma+\delta(\gamma))$*, where* $\Delta_{InitGuaranteed}(C) \leq Wd$*, for* $\rho = 0$*, and* $\Delta_{InitGuaranteed}(C) \leq W(\gamma+\delta(\gamma))$*, for* $\rho > 0$*.*

Lemma *ConvergenceTime – For* $\rho \geq 0$*, the convergence time is* $C = C_{Init} + \lceil \Delta_{Init}(C_{Init})/\gamma \rceil P$*.*

The Symbolic Model Verifier (SMV) was used in modeling of this protocol on a PC with 4GB of memory running Linux [SMV]. SMV allows the designers to formally verify temporal logic properties of finite state systems. SMV"s language description and modeling capability provide relatively easy translation from the pseudo-code. SMV also provides the desired capability to introduce randomness into the initial values of the variables.

The modeling in SMV consists of a global clock, *GlobalClock*, and a parameterized node, *Node*. The *GlobalClock* is used to measure the passage of time from the perspective of an external observer. The *Node* consists of local variables and executes the protocol. The synchronization properties are examined for a given network, where the network consists of a set of nodes that are instances of the *Node* module and are interconnected to reflect a desired topology.

Since in the protocol we do not limit K, model checking of all possible connected graphs for all K, even for idealized scenarios ($d = 0$, $\rho = 0$), is simply impossible. Model checking of all possible topologies for a given K is also a daunting task (Table 1). Given the limited resources available and to circumvent state space explosion, we had to limit the network size. Nevertheless, to verify our claims of the correctness of the protocol, we have model checked all possible graphs for smaller K. Additionally, we were able to model check some topologies for larger K. Table 3 is a list of the model checked networks with their sizes and corresponding number of topologies while bounding the drift to $\rho \leq 0.2$. Each row corresponds to a given K and two types of topologies considered with the number of model checked graphs of the possible total combinations for the corresponding topology type in its column.

The *Combo* topology is a 7-node graph consisting of a *Linear* topology of two nodes (1 and 2), a *Ring* topology of three nodes (2, 3, and 4), and a *Star* topology of four nodes (4, 5, 6, and 7) as depicted in Figure 5. Note that there is only one possible digraph for the *Linear* and *Star* topologies. Also, for three nodes, there are five digraphs (Table 1). However, for a *Ring* of three nodes, there are four variations. Therefore, after omitting symmetry, there are four digraphs for the *Combo* topology to be examined.

Table 3. Model checked networks.

K	**Topology** (all links bidirectional)	**Topology** (digraphs)
2	1 of 1	1 of 1
3	2 of 2	5 of 5
4	6 of 6	83 of 83
5	21 of 21	Single Directed Ring 2 Variations of Doubly Connected Directed Ring
6	112 of 112	-
7	Linear*	Linear*
7	Star*	Star*
7	Fully Connected*	Fully Connected*
7 (3×4)	Fully Connected Bipartite*	Fully Connected Bipartite*
7	Combo	4 of 4
7	Grid	-
7	Full Grid	-
9 (3×3)	Grid	-
15	Star*	Star*
20	Star*	Star*

* For *Linear* and *Star* topologies and for the network to be strongly connected (to be precise,1-connected), the links are by necessity bidirectional. For *Fully Connected (Complete)* and *Fully Connected Bipartite* topologies the links are by definition bidirectional.

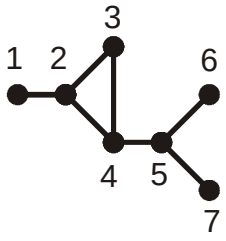

Figure 5. Combo topology.

Thus far, the model checking results have verified the correctness of the protocol as they apply to the networks with unidirectional and bidirectional links as described earlier (Section 2.3). In addition, the results so far confirm the claims of determinism and linear convergence. As a result, we conjecture that the protocol solves the general case of this problem for all $K \geq 1$.

5.1. Propositions

Computational tree logic (CTL), a temporal logic, is used to express properties of a system in this context. CTL uses atomic propositions as its building blocks to make statements about the states of a system. CTL then combines these propositions into formulas using logical and temporal operators with quantification over runs. In CTL formulas are composed of **path quantifiers**, *E* and *A*, and **temporal operators**, *X*, *F*, *G*, and *U* [Cla 1981].

Symbol	Meaning
E	there exists an execution
X	next
A	for all executions
F	finally (eventually)
G	globally
U	until

In this section the claims of convergence, closure, and congruence properties as well as the claims of maximum convergence time and determinism of the protocol are examined. Although in the description of the protocol convergence and closure properties are stated separately, they are examined via one CTL proposition. Validation of this general CTL proposition requires examination of a number of underlying propositions. In particular, since $\Delta_{LocalTimer}(t)$ is defined in terms of the *LocalTimer* of the nodes, examination of the properties that described proper behavior of the *LocalTimer* take precedence. In this section, the general propositions that verify the convergence, closure, and congruence properties of the protocol as well as the claims of maximum convergence time and determinism are examined followed by four supporting propositions.

The following properties are described with respect to only one node, namely *Node_1*. Since all nodes are identical, due to symmetry, the result of the propositions equally applies to other nodes. The variable *ElapsedTime* is used in some of these properties and is defined here.

ElapsedTime = (GlobalClock >= ConvergenceTime) ;

The *GlobalClock* is a measure of elapsed time from the beginning of the operation and with respect to the real time, i.e. external view. The *ElapsedTime* is indicative of the *GlobalClock* reaching its target maximum value of *ConvergenceTime*.

Proposition *ConvergenceAndClosure*: This proposition encompasses the criteria for the convergence and the closure properties as well as the claims of maximum convergence time and determinism. This proposition specifies whether or not the system will converge to the predicted precision after the elapse of convergence time, *ElapsedTime*, and whether or not it will remain within that precision thereafter. The expected result for this property is a true value.

```
AF (ElapsedTime) &                              -- Determinism Property
AG (ElapsedTime -> AllWithinPrecision) &        -- Convergence Property
AG ((ElapsedTime & AllWithinPrecision) ->
    AX (ElapsedTime & AllWithinPrecision))      -- Closure Property
```

The proper value of the *AllWithinPrecision* is determined by measuring the difference of maximum and minimum values of the *LocalTimers* of all nodes for the current tick and in conjunction with the result from the previous $(W+1)\gamma$ ticks. The expected difference of *LocalTimers* is the predicted precision bound.

The negation of the above proposition is listed below and the expected result is a false value. This property specifies that after the elapse of convergence time, *ElapsedTime*, whether or not the system will not converge or if it converges, whether or not it drifts apart beyond the expected precision bound.

```
AF (ElapsedTime) &
AG (ElapsedTime -> AllWithinPrecision) &
AG ((ElapsedTime & AllWithinPrecision) -> EX (!AllWithinPrecision))
```

Proposition *Congruence*: This property specifies the criteria for the congruence property of the protocol. The expected result for this property is a true value.

```
AF (ElapsedTime) &
AG ((ElapsedTime & (Node_1.LocalTimer= γ)) ->
    AX (ElapsedTime & AllWithinPrecision))      -- Congruence Property
```

The supporting properties follow.

Proposition 1: This property specifies whether or not time advances and the amount of time elapsed, *ElapsedTime*, has advanced beyond the predicted convergence time, *ConvergenceTime*. The expected result for this proposition is a true value.

```
AF (ElapsedTime)
```

Proposition 2: This property specifies whether or not the *LocalTimer* of a node takes on a given value in its range infinitely often, for instance, *P/2*. The expected result for this proposition is a true value.

```
AF (Node_1.LocalTimer = P/2)
```

Examining the negation of this property is expected to produce a false value. This proposition verifies that the *LocalTimer* of a node cannot never reach a given value.

```
EG !( Node_1.LocalTimer = P/2)
```

Proposition 3: This property specifies whether or not the *LocalTimer* of a node takes on all values in its range infinitely often. In other words, it verifies that the model does not deadlock. Furthermore, the value of the *LocalTimer* of a node at the next clock tick is different from its current value and is its expected next value in the sequence of 0 to *P*. The expected result for this proposition is a true value.

```
AG ((Node_1.LocalTimer = i) ->
  AX ((Node_1.LocaTimer= i) | (Node_1.LocalTimer = i+1))) &
AG ((Node_1.LocalTimer = P) ->
  AX (Node_1.LocalTimer = 0))
```

Examining the negation of this property is expected to produce a false value. This proposition verifies that the next value of the *LocalTimer* of a node cannot be the same as its current value. In other words, its value always advances within the expected range.

```
EG ((Node_1.LocalTimer = i) ->
  EX (Node_1.LocalTimer = i)) |
For all i = 0 .. (P-1)
```

Proposition 4: This property specifies whether or not the *LocalTimer* of a node takes on all values in its range infinitely often but beyond the convergence time, i.e. after *ElapsedTime* has become true. The expected result for this proposition is a true value. Examining the negation of this property is expected to produce a false value.

```
AF (ElapsedTime) &
AG (((ElapsedTime) & (Node_1.LocalTimer = i)) ->
  AX ((Node_1.LocalTimer= i) | (Node_1.LocalTimer = i+1))) &
AG (((ElapsedTime) & (Node_1.LocalTimer = P)) ->
  AX (Node_1.LocalTimer = 0))
```

6. Discussions

From the expression for $\Delta_{Init}(t)$ the synchronization time, C, and precision, π, are functions of the network topology and the drift rate, specifically, the graph‟s width and the amount of drift the network experiences. In other words, $C = f(W, \delta(P))$ and $\pi = f(W, \delta(P))$.

From the expressions for $\Delta_{Init}(t)$ and $\Delta_{InitGuaranteed}(t)$ it follows that for networks with small W values, $\Delta_{InitGuaranteed}(t)$ occurs instantaneously, but for networks with large W values $\Delta_{InitGuaranteed}(t)$ is a gradual process. The general equation for $\Delta_{Init}(t)$ applies to the ideal ($\rho = 0$, $d = 0$) and semi-ideal ($\rho = 0$, $d \geq 0$) scenarios. For these scenarios, $\Delta_{Init}(t) \leq W\gamma$.

Although the initial (coarse) synchrony, $\Delta_{Init}(t)$, occurs within C_{Init}, the initial guaranteed precision, $\Delta_{InitGuaranteed}(t)$, takes place after a number of periods and after achieving the initial synchrony. When $\rho >> 0$, i.e., $\delta(\gamma) \geq \gamma$, the equation for $\Delta_{InitGuaranteed}(t)$ is the same as $\Delta_{Init}(t)$, i.e., in this case, no improvement on $\Delta_{Init}(t)$ is achieved after the elapse time of C_{Init}. Recall that π is defined as the precision of the network over the duration of P while accounting for the overall drift in the network, i.e., $\pi = \Delta_{InitGuaranteed}(t) + \delta(P)$. When $\rho >> 0$ and since upon resynchronization process, $\Delta_{InitGuaranteed}(C) = \Delta_{Init}(C_{Init})$, no further improvement on $\Delta_{Init}(t)$ is achieved; therefore, no improvement on π can be guaranteed.

The general equation for π applies to the ideal ($\rho = 0$, $d = 0$) and semi-ideal ($\rho = 0$, $d \geq 0$) scenarios. Since $\Delta_{InitGuaranteed}(t) = f(W, \delta(P))$, for large values of $\delta(P)$, $\Delta_{InitGuaranteed}(C) = \Delta_{Init}(C_{Init})$ and no improvement on $\Delta_{Init}(t)$ is achievable. However, since typically $0 \leq \rho << 1$, for small values of $\delta(P)$, $\Delta_{InitGuaranteed}(C) < \Delta_{Init}(C_{Init})$ and improvement on $\Delta_{InitGuaranteed}(t)$ is possible. In particular, for the ideal and semi-ideal scenarios, subsequent resynchronization processes beyond the initial synchrony result in tighter precision. Specifically, for $C = C_{Init} + \lceil \Delta_{Init}(C_{Init}) / \gamma \rceil P$, for the ideal scenario, the result is $\Delta_{InitGuaranteed}(C) = 0$ and $\pi = 0$, while for the semi-ideal scenario, $\Delta_{InitGuaranteed}(t) = Wd$ and $\pi = Wd$.

Therefore, $\Delta_{InitGuaranteed}(C)$ is 0, Wd, and $W(\gamma + \delta(\gamma))$, for the ideal, semi-ideal, and realizable systems ($\rho \geq 0$, $d \geq 0$), respectively. After synchrony, for the ideal scenario, the nodes periodically pulsate in perfect unison (true synchrony). For the semi-ideal scenario, even in the absence of drift, the system‟s behavior resembles a ripple effect where the nodes remain at most one d apart from each other with the leading node as the center and originator of the *ripple*. Also, for realizable systems, due to the effects of drift, the system‟s behavior resembles a ripple effect. However, when the nodes periodically pulsate, depending on the amount of drift, the lights emanate at at most one γ apart from each other with the leading node as the center and originator of the *ripple*.

In this report we have studied the system in the absence and presence of ρ. In Section 6.2 we discuss whether or not ρ should be bounded and determine its theoretical upper bound. Recall that $\pi = f(W, \delta(\gamma))$ and $C = f(W, \delta(\gamma))$. Therefore, depending on the values of W and $\delta(\gamma)$, the precision of the network and the convergence time may be quite large. So, is it possible to achieve faster synchrony? Is it possible to achieve a desired precision? From the expression for π it follows that for networks with small W values, synchronization occurs instantaneously with

optimal precision while for networks with large W values, synchronization is a gradual process and with larger precision. For instance, for a fully connected graph, $W = 1$, $\pi = d+\delta(\gamma)$ is at its minimum with minimal dependence on the drift, and the convergence time is at its minimum value of $C = C_{Init}$, whereas for a linear graph, $W = K - 1$, π is at its maximum and more dependent on the drift, and the convergence time is at its maximum value of C. Indeed, for the worst case where drift is very high, no improvement on $\Delta_{Init}(t)$ is possible no matter how much time passes. So, to achieve a desired precision, we must reduce either W or $\delta(P)$, or both.

To reduce W, we have to add new links to the graph, but where to add the new links and how many links to add? The idea of adding a few random links and rewiring links with a certain probability to provide shortcuts between different segments of a graph has been studied by Watts and Strogatz [Wat 1998] and others [Gad 2000] [Bar 2002] [Hon 2002, 2004] [Li 2004] [Gom 2007]. As Arenas [Are 2008] concluded from these studies, "In general, the addition of shortcuts to regular lattices improves synchronization." and "The basic observation is that the network synchronizes when the coupling strength is increased." These studies have shown the effects of adding new links, but they do not specify how many links and where to add them in order to expedite synchronization. However, thus far in our report we have established that $\pi = f(T, \rho)$ and, so, $\pi = f(W, \delta(P))$. Therefore, to achieve the tightest precision, i.e., $\pi = d+\delta(\gamma)$, we need to add new links to the graph such that we successively halve the graph width W and, hence, double the precision. This implies that the number of links (or edges) to be added, E, is given by $E \geq \lceil \log_2 {}^{\Delta_{Init}(t)} \rceil$.

To reduce the drift, more accurate oscillators are needed, but the more accurate the oscillators, the higher the cost. What if the graph cannot or should not be modified by adding new links? Also, there are no perfect oscillators. So, what if we cannot improve upon the drift beyond a practical limit? Is there another way to achieve synchrony faster and with more accurate precision? The following section addresses these issues and examines variations of this protocol.

6.1. Variations Of The Synchronization Protocol

In this section we present several variations of the synchronization protocol. But first we provide an intuitive explanation. One of the key elements of the presented protocol is the proper setting of the *LocalTimer* upon receiving a *Sync* message. In the protocol we set the *LocalTimer* to γ. The rational is that when a node times-out, it resets its *LocalTimer*, i.e., *LocalTimer* = 0, and after one γ, the transmitting and receiving nodes would naturally be in relative synchrony of at most d clock ticks from each other. If we set the *LocalTimer* to D, the protocol behaves similarly but with a lower precision. In fact, as we"ll see in the following section, setting the *LocalTimer* to any value less than γ produces lower precision than setting it to γ. We will not consider setting the *LocalTimer* to D a variation of the protocol.

Setting the *LocalTimer* to other values would not produce the desired effect. On the other hand, if a node gets interrupted, the receiving nodes have no knowledge of the transmitting node"s *LocalTimer* value (which could be either 0 or γ). Once again, in the protocol we chose to set the *LocalTimer* to γ upon interrupt and we verified that it achieves the desired goal. However, upon interrupt, the *LocalTimer* could be assigned to other values, but what value should be chosen?

An arbitrary value is not going to produce the desired synchrony, but if the value of the transmitting node"s *LocalTimer* is forwarded, then the *LocalTimer* of the receiving node could be set to that value (offset by γ) and once again the two nodes will be in relative synchrony. In the following sections, we analyze these variations. We believe that transmitting any value other than the transmitting node"s *LocalTimer* value does not produce the desired effect.

6.1.1. Variation #1, Reset

This variation is depicted in Figure 6 where *LocalTimer* is reset, i.e., *LocalTimer* = 0, upon receiving a *Sync* message (statements *E1* and *E2*).

Synchronizer:
E1: if ((Message = *Sync*) and (*LocalTimer* < *D*))
    LocalTimer := 0,
E2: elseif (((Message = *Sync*) and (*LocalTimer* ≥ T_S))
    LocalTimer := 0,
    Transmit *Sync*,
E3: elseif (*LocalTimer* ≥ *P*)
    LocalTimer := 0,
    Transmit *Sync*,
E4: else
    LocalTimer := *LocalTimer* + 1.

Figure 6. The synchronizer for variation #1, Reset.

Thus far, the model checking results have verified the correctness of this variation of the protocol. This variation of the protocol also synchronizes the network for $\rho \geq 0$ and $d \geq 0$ with the same $\Delta_{Init}(t)$, i.e., $\Delta_{Init}(C_{Init}) \leq (K - 1)(\gamma+\delta(\gamma))$. Also, when $\rho = 0$ and $d = 0$, unlike the original protocol where $\Delta_{InitGuaranteed}(t) = 0$, $\Delta_{InitGuaranteed}(t) = W\gamma$. Setting the *LocalTimer* to other values between 0 and γ would produce similar results as the original protocol and this variation of it with $0 < \Delta_{InitGuaranteed}(t) < W\gamma$.

In this version, since $\Delta_{InitGuaranteed}(t) = W\gamma$, even in the absence of drift, the system"s behavior resembles a ripple effect where the nodes remain at most one γ apart from each other with the leading node as the center and originator of the *ripple*.

From variation #1 and the original protocol, one could conclude that upon receiving a *Sync* message, setting the *LocalTimer* from 0 to γ results in improvement of the initial guaranteed precision. An interesting question is whether setting the *LocalTimer* to a greater value than γ would improve upon the performance even further. As argued in the opening of this section, the next logical value beyond γ would be *LocalTimer* of the transmitting node. The following variation of the protocol is based on this idea.

6.1.2. Variation #2, Jump Ahead

This variation is depicted in Figure 7. In this variation, the current value of the *LocalTimer* is transmitted along with the *Sync* message and, so, upon receiving a *Sync* message *LocalTimer* is set to the incoming value plus γ to compensate for the worst case message delay. If the sum reaches or exceeds *P*, the *LocalTimer* is reset to zero (statements *E1* and *E2*).

Synchronizer:
E1: if ((Message = *Sync*) and (*LocalTimer* < *D*))
LocalTimer := *LocalTimerIn* + γ,
if (*LocalTimer* ≥ *P*)
LocalTimer := 0,
E2: elseif (((Message = *Sync*) and (*LocalTimer* ≥ T_S))
LocalTimer := *LocalTimerIn* + γ,
if (*LocalTimer* ≥ *P*)
LocalTimer := 0,
Transmit *Sync* and *LocalTimer*,
E3: elseif (*LocalTimer* ≥ *P*)
LocalTimer := 0,
Transmit *Sync* and *LocalTimer*,
E4: else
LocalTimer := *LocalTimer* + 1.

Figure 7. The synchronizer for variation #2, Jump Ahead.

We do not provide a detailed proof for this variation. However, thus far, partial model checking has also confirmed the correctness of this variation of the protocol. Nevertheless, in this section we state the lemmas and theorems that provide specific measures to this variation of the protocol.

Lemma *InitGuaranteed – For all* $t \geq C$, *the initial guaranteed precision is given by* $\Delta_{InitGuaranteed}(t) = (1+d)\delta(P)$.

Theorem *Congruence – For all nodes* N_i *and for all* $t \geq C$, (N_i.*LocalTimer*$(t) = W\gamma$ implies $\Delta_{Net}(t) \leq \pi$).

This variation introduces more overhead due to the transmission of *LocalTimer* value but synchronizes the network for $\rho \geq 0$ and $d \geq 0$ with the same initial precision. In other words, $\Delta_{Init}(C_{init}) \leq (K - 1)(\gamma+\delta(\gamma))$. However, this variation produces tighter initial guaranteed precision for the same convergence time, i.e., $\Delta_{InitGuaranteed}(C) = (1+d)\delta(P)$ and $C = C_{Init} + \lceil \Delta_{Init}(C_{init}) / \gamma \rceil P$.

This variation of the protocol has two drawbacks. The first drawback is that it requires greater number of exchanges of *Sync* messages during the convergence process. The excess transmission of the *Sync* messages is due to the burst of relays of *Sync* messages prior to the

convergence. Note that since after receiving a *Sync* message the *LocalTimer* of a node gets incremented, all messages will eventually die out when the *LocalTimer* of a node reaches or exceeds its maximum value of P. Recall that in the original protocol, by setting the *LocalTimer* to γ, the node immediately enters the ignore window, a time interval where it ignores all incoming *Sync* messages. In this variation, however, depending on the initial value of the *LocalTimer* of a node, a message may not get ignored until eventually the *LocalTimer* of a node reaches or exceeds its maximum value of P and then enters the ignore window.

The second drawback is that due to an interrupt, the slowest nodes may never get set to a γ during a resynchronization process even when the system is in synchrony. As a result (Theorem *Congruence*), for $t \geq C$, the nodes are in synchrony when $N_i.LocalTimer(t) = W\gamma$. In the original protocol, the following is the expected range of values of the *LocalTimer* of a node.

$N_i.LocalTimer = [[0 \,..\, \gamma] \,..\, [(P - \delta(P)) \,..\, P]]$

However, for this variation the following is the case.

$N_i.LocalTimer = [[0 \,..\, W\gamma] \,..\, [(P - \delta(P)) \,..\, P]]$

Where, $[X1 \,..\, X2]$ mean that the node can take on any value in the range of $X1$ to $X2$. Therefore, for this variation of the protocol, the guaranteed effective interval of a node is as follows.

$N_i.LocalTimer = [W\gamma \,..\, (P - \delta(P))]$

6.2. Bound On The Drift Rate, ρ

Thus far, we have seen the effects of the drift rate, ρ, on the convergence time and guaranteed precision of the network. We have seen that, in its absence, the best possible precision is achieved in the shortest amount of time and how its presence decreases the precision and increases the convergence time. In this section we discuss the role of ρ even further.

Generally, $0 \leq \rho << 1$. However, in the proofs we had considered $0 \leq \rho < 1$. In this section we examine the following questions. Should the drift rate be bounded? If so, what should its upper bound be? What are the ramifications of limiting or not limiting the drift in a network? Is there a threshold?

Typically and in practical applications, oscillator drift rate, ρ, is bounded to 100 parts per million, i.e., $\rho \leq 10^{-4}$. In other words, $\lambda = (1+\rho) - 1/(1+\rho) \leq 0.0002$. Recall that for bidirectional topologies, $P = f(K, \rho)$. As ρ increases so does P and $\delta(P)$. The implication of $\rho > 0$ is that the effective duration of the slowest node is $P - \delta(P)$, i.e., when the fastest node reaches P, the slowest node is at $P - \delta(P)$. If the fastest and slowest nodes happen to be adjacent to each other, then the slowest node gets interrupted by the fastest node and never goes beyond $P - \delta(P) + \gamma$. Let us restrict ρ such that the slowest node will always be at above 50% of the fastest node, i.e., $P - \delta(P) > \frac{1}{2}P$. This implies that the value of $\delta(P)$ should not exceeds $\frac{1}{2}P$, i.e., the desired condition is for $\lambda < \frac{1}{2}$.

Table 4. Various ρ and its corresponding λ.

ρ	λ
0.0001	0.0002
0.1	0.191
0.2	0.367
0.3	0.531
0.4	0.686
0.5	0.833
0.6	0.975
0.7	1.112

Table 4 provides a list of various values for ρ and its corresponding λ. From this table, there is a threshold for ρ (at about 0.3) where $\lambda > ½$. At and beyond this threshold, as the slowest nodes reach ½*P*, the fastest nodes will have reached *P* and timed-out.

Although the protocol works and achieves synchrony even for large ρ, i.e., $\rho < 1$, for practical applications a large value for ρ is not realistic. In other words, for large ρ, besides the substantial decrease in the network precision, the slowest nodes never get to do much meaningful work because their activity duration is substantially reduced as they are routinely timed-out by the fastest nodes.

6.3. Directed Graphs And Dynamic Graphs

We have elaborated thus far in previous sections that the general form of the distributed synchronization problem, *S*, is defined by the following septuple.

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

In other words, the distributed synchronization problem is a function of the number of nodes, network topology, communication delay, communication imprecision, oscillator drift rate, synchronization period, and number of faults, respectively.

However, so far, we have considered topologies with static nodes and links. This restriction helped to reduce the complexity of the problem to a more manageable size. We now define the most general form of the distributed synchronization problem, *S'*, by the following septuple.

$$S' = (K(t), T(t), D, d, \rho, P, F)$$

Where, *K(t)* represents the **dynamic node count** and *T(t)* represents the **dynamic topology** for a given *K(t)*.

In a dynamic node count, the number of nodes comprising the network can change at any time. Since the nodes are anonymous and do not have unique identifiers, the presented protocol and its variations are readily applicable to this scenario, provided that the new nodes enter the network from a reset state where they are clear of all residual effects.

The dynamic topology allows for topologies with any combination of unidirectional and bidirectional links as described in Section 2.3, whether they are static or dynamic. In other words, for a given *K(t)*, the number of links can change at any time.

We have model checked a number of topologies with static nodes and various combinations of static unidirectional and bidirectional links and, thus far, the model checking results have verified the correctness of the protocol. We conjecture that the presented protocols are applicable to the general case.

7. Conclusions

How can a distributed system solve a problem that is inherently global by executing a set of rules locally? In this report, we have attempted to answer this question by providing a solution that synchronizes an arbitrary digraph, ranging from fully connected to 1-connected networks of nodes, under variety of conditions ranging from ideal to non-ideal circumstances. These networks include grid, ring, fully connected, bipartite, and star (hub) formation, to name a few, while allowing differences in the network elements. In our proposed solution, there is no central control or a centrally generated signal, pulse, or message. Nodes are anonymous, i.e., they do not have unique identities. We discussed the complexity of the problem and defined the parameters constituting the distributed synchronization problem.

We provided an intuitive description of the behavior of the protocol. We also provided an outline of a deductive proof of the protocol followed by the model checking results that have verified the correctness of the protocol as they apply to the networks with unidirectional and bidirectional links. In addition, the model checking results so far have confirmed the claims of determinism and linear convergence. We also provided variations of the protocol and presented model checking results of those variations. We also discussed generalization of the protocol to include dynamic node count and dynamic topology. Details of the deductive proof and details of the model checking efforts of this protocol, and its variations, are the subject of subsequent reports.

We elaborated on the effects of the oscillator drift rate on the convergence time and network precision and discussed whether or not it should have an upper bound.

The proposed self-stabilizing protocol is expected to have many practical applications as well as many theoretical implications. Embedded systems, power grid, distributed process control, synchronization, computer networks, the Internet, Internet applications, security, safety, automotive, aircraft, distributed air traffic management systems, swarm systems, wired and wireless telecommunications, graph theoretic problems, leader election, time division multiple access (TDMA), and the SPIDER[2] project [Tor 2005A, 2005B] at NASA-LaRC are a few examples. These are some of the many areas of distributed systems that can use synchronization in order to design more robust distributed systems.

There does not seem to be a consensus on the definition of either emergent behavior or emergent systems [Cor 2002]. However, in the context of self-organization systems Goldstein defines emergence as: "the arising of novel and coherent structures, patterns and properties during the process of self-organization in complex systems" [Gol 1999]. Emergent systems tend to display a collective behavior that is greater than the sum of their parts. An emergent behavior or emergent property surfaces in systems as a result of the interactions at an elemental level. In other words, an emergent property of a system is one that is not a property of any component of that system, but is still a feature of the system as a whole. The family of clock synchronization protocols presented in this report is an emergent system. In these protocols all nodes operate asynchronously while the system operates synchronously. The local physical oscillators of the

[2] Scalable Processor-Independent Design for Enhanced Reliability (SPIDER).

nodes are and remain asynchronous while the system synchronizes at a higher level. These protocols are designed to be deterministic and analyzable.

Finally, we believe this protocol can be used as a basis for modeling and studying mass synchrony as exhibited in biological and social systems. For instance, based on personal observations, we believe the first variation of the protocol (*Reset*) is closer to the observed behavior of frogs and fireflies.

References

[Are 2008] Arenas, A.; Diaz-Guilera, A.; Kurths, J.; Moreno, Y.; Zhou, C.: "Synchronization in complex networks," PACS: 05.45.Xt, 89.75.Fb, 89.75.Hc, December 2008.

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

[Bre 2008] Brede, M.: "Locals vs. global synchronization in networks of non-identical kuramoto oscillators," Europ. Phys. J. B 62 (2008) 87–94.

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

[Cor 2002] Corning, P.A.: "The Re-Emergence of "Emergence": a Venerable Concept in Search of a Theory," Complexity 7(6): pp. 18-30, 2002.

[Cla 1981] Clarke, E.M.; Emerson, E.A.: Design and synthesis of synchronization skeletons using branching time temporal logic. In *Logic of Programs: Workshop, Yorktown Heights, NY, May 1981, LNCS* 131. Springer, 1981.

[Dal 2003] Daliot, A.; Dolev, D.; Parnas, H.: "Linear Time Byzantine Self-Stabilizing Clock Synchronization," Proceedings of 7th International Conference on Principles of Distributed Systems (OPODIS-2003), La Martinique, France, December 2003.

[Dav 1978] Davies, D.; Wakerly, J.F.: "Synchronization and matching in redundant systems," IEEE Transactions on Computers, 27(6), pp. 531-539, June 1978.

[Dij 1974] Dijkstra, E.W.: "Self stabilizing systems in spite of distributed control," Commun. ACM 17, pp. 643-644, 1974.

[Dol 2004] Dolev, S.; Welch, J.L.: "Self-Stabilizing Clock Synchronization in the Presence of Byzantine Faults," Journal of the ACM, Vol.51, No. 5, pp. 780-799, September 2004.

[Ear 2003] Earl, M.G.; Strogatz, S.H.: "Synchronization in Oscillator Networks With Delayed Coupling: A Stability Criterion," The American Physical Society, 2003.

[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.

[Gle 2006] Gleiser, P.M.; Zanette, D.H.: "Synchronization and structure in an adaptive oscillator network," Europ. Phys. J. B 53 (2006) 233–238.

[Gol 1999] Goldstein, j: "Emergence as a Construct: History and Issues," Emergence 11, pp. 49-72, 1999.

[Gom 2007] G´omez-Garde˜nes, 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.

[Hud 1918] George H. Hudson, Science 48, pp. 573-575, 1918.

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

[Lam 1982] Lamport, L.; Shostak, R.; Pease, M.: "The Byzantine General Problem," ACM Transactions on Programming Languages and Systems, 4(3), pp. 382-401, July 1982.

[Lam 1985] Lamport, L; Melliar-Smith, P.M.: "Synchronizing clocks in the presence of faults," J. ACM, vol. 32, no. 1, pp. 52-78, 1985.

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

[Lis 1970] Liskovets, V.A.: "number of strongly connected directed graphs," Matmaticheskie Zameki, Vol. 8, No. 6, pp. 721-732, December 1970.

[Mal 2006A] Malekpour, M.R.; Siminiceanu, R.: "Comments on the „Byzantine Self-Stabilizing Pulse Synchronization" Protocol: Counterexamples." NASA/TM-2006-213951, February 2006.

[Mal 2006B] 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 2008] Malekpour, M.R.: "Verification of a Byzantine-Fault-Tolerant Self-Stabilizing Protocol for Clock Synchronization." IEEE Aerospace Conference, March 2008.

[Mal 2009] Malekpour, M.R.: "A Self-Stabilizing Byzantine-Fault-Tolerant Clock Synchronization Protocol," NASA/TM-2009-215758, June 2009.

[Min 2002] Miner, P.S. ; Malekpour, M.R.; Torres, W.: "A Conceptual Design For a Reliable Optical Bus (ROBUS)", Presented at the 21st Digital Avionics Systems Conference (DASC), Irvine, California, October 27-31, 2002.

[Mir 1990] Mirollo, R.E.; Strogatz, S.H.: "Synchronization of Pulse-Coupled Biological Oscillators," SIAM Journal on Applied Mathematics, Vol. 50, No. 6, pp. 1645-1662, December 1990.

[Nis 2006A] Nishikawa, T.; Motter, A.E.: "Maximum performance at minimum cost in network synchronization," Physica D 224 (2006) 77–89.

[Nis 2006B] Nishikawa, T.; Motter, A.E.: "Synchronization is optimal in nondiagonalizable networks," Phys. Rev. E 73 (2006) 065106.

[Pes 1975] Peskin C.: "Mathematical Aspects of Heart Physiology", 1975.
http://www.math.nyu.edu/faculty/peskin/heartnotes/index.html
http://www.math.nyu.edu/faculty/peskin/heartnotes/CLN-Peskin1975-7.pdf

[Sloane] Sloane, N.J.A.: Sequence A001349. The On-Line Encyclopedia of Integer Sequences.
http://www.research.att.com/projects/OEIS?Anum=A001349,
http://mathworld.wolfram.com/ConnectedGraph.html.

[SMV] http://www-2.cs.cmu.edu/~modelcheck/smv.html

[Sri 1987] Srikanth, T.K.; Toueg, S.: "Optimal clock synchronization," Journal of the ACM, 34(3), pp. 626–645, July 1987.

[Str 2003] Strogatz, S.H.: "SYNC, How Order Emerges From Chaos in the Universe, Nature, and Daily Life," ISBN 978-0-7868-8721-7, 2003.

[Tor 2005A] Torres-Pomales, W.; Malekpour, M.R.; Miner, P.S.: "ROBUS-2: A Fault-Tolerant Broadcast Communication System," NASA/TM-2005-213540, March 2005.

[Tor 2005B] Torres-Pomales, W.; Malekpour, M.R.; Miner, P.S.: "Design of the Protocol Processor for the ROBUS-2 Communication System," NASA/TM-2005-213934, pp. 252, November 2005.

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

[Wel 1988] Welch, J.L.; Lynch, N.: "A New Fault-Tolerant Algorithm for Clock Synchronization," Information and Computation volume 77, number 1, pp.1-36, April 1988.

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 i^{th} node
M_i	the i^{th} 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
$\Delta_{ij}(t)$	precision between *LocalTimers* of any two adjacent nodes N_i and N_j at time t
$\Delta_{Init}(t)$	initial precision among *LocalTimers* of all nodes at time t
$\Delta_{InitGuaranteed}(t)$	initial guaranteed precision among *LocalTimers* of all nodes at time t
$\delta(t)$	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)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
01- 02 - 2011	Technical Memorandum	

4. TITLE AND SUBTITLE
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-19976

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-2011-217054

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

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This report presents a self-stabilizing distributed clock synchronization protocol in the absence of faults in the system. 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, other than the presence of at least one node, 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 an outline of a deductive proof of the correctness of the protocol. A model of the protocol was mechanically verified using the Symbolic Model Verifier (SMV) for a variety of topologies. Results of the mechanical proof of the correctness of the protocol are provided. The model checking results have verified the correctness of the protocol as they apply to the networks with unidirectional and bidirectional links. In addition, the results confirm the claims of determinism and linear convergence. As a result, we conjecture that the protocol solves the general case of this problem. We also present several variations of the protocol and discuss that this synchronization protocol is indeed an emergent system.

15. SUBJECT TERMS
Algorithm; Clock Synchronization; Communication Network; Digraphs; Distributed Systems; 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	42	(443) 757-5802