NASA/TM-2017-219638

An Autonomous Distributed Fault-Tolerant Local Positioning System

Mahyar R. Malekpour
Langley Research Center, Hampton, Virginia

July 2017

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 NTRS Registered and its public interface, the NASA Technical Reports 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 counter-part of peer-reviewed formal professional papers but has 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 organizing and publishing research results, distributing specialized research announcements and feeds, providing information desk and personal search support, and enabling data exchange services.

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 to help@sti.nasa.gov

- Phone the NASA STI Information Desk at 757-864-9658

- Write to:
  NASA STI Information Desk
  Mail Stop 148
  NASA Langley Research Center
  Hampton, VA 23681-2199

NASA/TM-2017-219638

An Autonomous Distributed Fault-Tolerant Local Positioning System

Mahyar R. Malekpour
Langley Research Center, Hampton, Virginia

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-2199

July 2017

Available from:

NASA STI Program / Mail Stop 148
NASA Langley Research Center
Hampton, VA 23681-2199
Fax: 757-864-6500

Abstract

We describe a fault-tolerant, GPS-independent (Global Positioning System) distributed autonomous positioning system for static/mobile objects and present solutions for providing highly-accurate geo-location data for the static/mobile objects in dynamic environments. The reliability and accuracy of a positioning system fundamentally depends on two factors; its timeliness in broadcasting signals and the knowledge of its geometry, i.e., locations and distances of the beacons. Existing distributed positioning systems either synchronize to a common external source like GPS or establish their own time synchrony using a scheme similar to a master-slave by designating a particular beacon as the master and other beacons synchronize to it, resulting in a single point of failure. Another drawback of existing positioning systems is their lack of addressing various fault manifestations, in particular, communication link failures, which, as in wireless networks, are increasingly dominating the process failures and are typically transient and mobile, in the sense that they typically affect different messages to/from different processes over time. We solve this problem by first employing fault-tolerant distributed clock synchronization protocols to achieve the theoretical synchrony of one-clock tick across the distributed system of nodes (beacons) and, consequently, determine the geometry of the network, and then use trilateration to accurately determine the current location of the intended object.

Keywords: GPS, independent, positioning system, autonomous, geo-location, dynamic, mobile, fault-tolerant, synchronization, distributed

Table of Contents

1. Introduction

The Global Positioning System (GPS) is a versatile, generally available worldwide navigational system based on the reception of signals from a network of satellites orbiting the globe. GPS is widely used as an external source for geo-location in 2D and 3D Local Positioning Systems (LPS), e.g., automobile navigation and unmanned aerial vehicles (UAV). The growing reliance upon a single source of positioning information introduces a significant vulnerability. Since GPS is a weak signal, it is vulnerable to jamming. Furthermore, the multitude of satellites forming GPS was intended to provide accurate signals and not necessarily support a degree of fault-tolerance, and so GPS is vulnerable to spoofing (i.e., the purposeful sending of inaccurate data to the receiver). The European Union (Galileo), Russia (GLONASS), China (BeiDou) and other countries have deployed and are in the process of deploying more satellites in order to improve the accuracy and reliability of Global Navigation Satellite Systems (GNSSs). Nevertheless, as Rizos [Riz 2013] pointed out, "the most severe limitation of GNSS performance will still remain; the accuracy of positioning deteriorates very rapidly when the user receiver loses direct view of the satellites, which typically occurs indoors, or in severely obstructed urban environments, steep terrain and in deep open-cut mines." As a result, LPS are often used in either a complementary or alternative fashion to the satellite systems, especially in areas where GPS signals are sufficiently degraded. Although Long-range LPS, e.g., Decca Navigator System and LORAN (LOng RAnge Navigation), have been used for navigation of ships and aircraft, nowadays, a LPS typically refers to a system with limited range used for outdoor and/or indoor applications.

Positioning systems that consists of a network of three or more signaling beacons have been used for navigation and surveying by providing location information within the coverage area. The reliability and accuracy of such a positioning system fundamentally depends on two factors; first, its timeliness in broadcasting signals, i.e., whether or not the signals are transmitted at the same time or as close to the same time as possible, and second, the knowledge of its geometry, i.e., locations and distances of its beacons. The more accurate the time at each beacon and the higher the precision across the network, the more accurate the estimated position at the receivers. Similarly, the more accurate the geometry and knowledge of the location and distances of the beacons from each other, the more accurate the estimated position at the receivers will be.

Distributed LPSs (DLPS) either synchronize to a common external source like GPS or establish their time synchrony internally. GPS satellites operate on very high precision atomic clocks that "tick" with an accuracy of one nanosecond (providing position accuracy within 5 to 10 meters) and are synchronized to the Coordinated Universal Time (UTC), which is the primary time standard by which the world regulates clocks and time [GPS]. The atomic clocks on these satellites are very accurate (drift rate 10^{-13}, or less) and very expensive, thus, cost prohibitive for most applications. LPSs on the other hand operate with lower quality clocks, i.e., have a higher drift rate, that are more affordable. These clocks, however, need to be periodically resynchronized to account for their inherent drift and far more frequently than the atomic clocks.

In DLPSs that use master-slave scheme to internally establish their time synchrony, typically, a particular beacon is designated as the master and other beacons synchronize to it, e.g., the Locata system [Riz 2013]. Due to its centralized nature, a network with master-slave scheme results in a single point of failure. Another drawback of existing DLPSs is their lack of addressing various

fault manifestations, in particular, communication link failures. In [Bie 2011] Biely *et al.* make the following two points; first, due to the high reliability of modern processors, communication-related failures like receiver overruns (run out of buffers), unrecognized packets (synchronization errors), and CRC errors (data reception problems) in all sorts of wireless networks are increasingly dominating process failures, and second, such link failures are typically transient and mobile, in the sense that they typically affect different messages to/from different processes over time.

The geo-location and time-synchrony problems have a lot in common. Geo-location requires a distributed system of at least four beacons to estimate the location in 3-dimensions [GPS]. The fourth beacon is necessary to account for discrepancies in value of time readings, $((x, y, z), t)$, which is primarily due to low quality of the receiver's local oscillator. Similarly, the time-synchrony problem requires a minimum of four nodes (beacons) to tolerate a malicious faulty behavior [Lam 1982, 1985][Dol 1984]. The capability of a distributed network of beacons to autonomously self-synchronize and, subsequently, provide reliable signaling for proper geo-location, at the receivers and independent of GPS, is essential in reducing total reliance on an external source like GPS. The internal timing of an independently self-synchronizing network can readily be realigned to GPS time when it becomes available. Similarly, prior knowledge of the exact locations of the beacons and their network geometry is not necessary; an autonomous distributed fault-tolerant local positioning system should be able to first determine its own geometry and then realign it to the world map when GPS data is available.

Thus, our research goals were to provide a highly reliable, GPS-independent, fault-tolerant, redundant system for geo-location of UAVs for various projects within the center and across the agency, such as NASA's Unmanned Aircraft Systems (UAS) Integration in the National Airspace System (NAS) Project and Langley Research Center's CERTAIN (City Environment for Range Testing of Autonomous Integrated Navigation). Our goals also included a DLPS suitable for high-dynamic systems by accommodating capabilities for UAVs to maneuver at high speed, and in dynamic and mobile environments. Such a DLPS has to be able to autonomously and continually establish its time-synchrony and determine its geometry independent of an external source. NASA views autonomy as distinct from automation in accordance with its space initiatives. Automated systems provide control or execution of a system without human intervention or commands. This does not preclude the possibility of operator input, but such input is explicitly not required for an automated function [Con 2006][Hay 2014, 2015].

We use the terms protocol and algorithm interchangeably. We also use the terms *nodes* and *object* generically as the protocols discussed in this paper are applicable to various applications. For geo-location applications, however, the terms *nodes* and *object* refer to *beacons* and *vehicle* (or target), respectively.

We present our research results and solutions to the autonomous distributed positioning problem. We solve this problem by first employing distributed clock synchronization protocols to achieve the theoretical synchrony of one-clock tick across the distributed system of nodes and, consequently, determine the geometry of the network, and then use trilateration[*] to accurately

[*] Trilateration is the process of determining absolute or relative location of a point given a set of sphere centers, their locations, and their radii, using the geometry of circles, spheres or triangles.

determine the current location of the intended object. Achieving fine synchrony is in turn a two-step process using two complementary protocols. First, we use a primary distributed clock synchronization protocol to establish *coarse synchrony* across the distributed system of nodes. Second, we use a secondary protocol to achieve *fine synchrony*, which is a theoretical limit bounded to one clock tick across the system.

Based on fault-tolerance requirements, we employ one of the two mechanically verified distributed clock synchronization algorithms to establish coarse synchrony across the nodes, namely, the Digraph Protocol [Mal 2011] that handles detectable faults and is versatile, in terms of the variety of topologies it is applicable to, and the Symmetric-Fault-Tolerant Protocol [Mal 2015] which, as its name implies, tolerates symmetric faults. A fault is symmetric when all good nodes observe consistent error manifestations, but do not know that it is bad [Min 2002]. These protocols guarantee synchrony, with an initial precision of $f(\gamma)$, where γ is the maximum communication delay between any two nodes, across a distributed system of nodes, from a random start and in the presence of their representative fault types. Since γ can be greater than one clock tick, this achieves coarse synchrony. Once coarse synchrony is achieved, a secondary algorithm subsequently attains the theoretical precision of one clock tick. Once the fine synchrony among the nodes is achieved, geometry of the network is determined, and location of the object is estimated, at the object and/or at the nodes, using trilateration. We would like to point out that this report is not about developing a new physical layer for wireless communication. We believe, however, that the technological knowledge and capabilities do exist for developing a system that meets the physical layer requirements of the ideas and solutions described here.

This report is organized as follows. In Section 2 we describe the fault models. In Section 3 we provide a system overview. We present the Primary and Secondary clock synchronization protocols in Sections 4 and 5, respectively. In Section 6 we present locating an object in static and dynamic environments. In Section 7 we present an algorithm integrating the Primary and Secondary protocols. Finally, we present concluding remarks in Section 8.

2. Fault Models

A distributed system is *synchronous* if there is a known upper bound on the transmission delay of messages and if there is a known upper bound on the processing time of a piece of code. In synchronous message-based distributed systems, a fault is typically defined as a message that was not transmitted when it was expected, a message that was transmitted but not received, or received but not accepted, i.e., deemed invalid by a receiver. Consequently, there are two viewpoints, in the *node-fault model*, the faults are associated with the source node of the message and all fault manifestations between the source and the destination nodes for the messages from that source count as a single fault, which is specially the case when the faults are associated with an arbitrary (Byzantine) faulty node [Lam 1982][Dri 2003][Min 2004]. In this model all links are assumed to be good. Miner *et al.* [Min 2004] for instance, model the absence of a link as a link fault and even though both nodes and links failures are considered, they abstractly model link failures as failures of the source node.

In the *link-fault model*, a fault is associated with the communication means connecting the source node to the destination node. In this model, all nodes are assumed to be good and an invalid message at the receiving node is counted as a single fault for the corresponding input link. Thus, from the global perspective, a Byzantine faulty node manifests as one or more link failures.

A link-fault model introduced by Schmid *et al.* [Sch 2002] is called *perception-based hybrid fault model*, where faults are viewed from the perspective of the receiving nodes. Faults are associated with their input links, and all nodes are assumed to be good. They argued that since F faulty nodes can produce at most F faulty perceptions in any node, the link-fault model is compatible with the traditional node-fault model and so, all existing lower bound and impossibility results remain valid.

In this report, we consider both types of faults. To protect against spoofing, we consider *node-fault model*, where a node is detectable faulty, symmetric faulty, or Byzantine (arbitrary) faulty, provided the fault manifests itself to at most F other nodes, or experiences no more than F faults if it is a good node, where $F \geq 0$. To tolerate sporadic link failures, we also consider the *link-fault model*, where the nodes are assumed to be good and, at any round, no more than F link faults are perceived at a receiving good node. As described in [Mal 2017], for both types of faults, we assume that the maximum number of simultaneous faults in the network is limited to F.

3. System Overview

We consider synchronous message-passing wireless distributed systems and model the system as a graph with a set of nodes (vertices) that communicate with each other by sending messages via a set of wireless communication links (edges) that represent the nodes' interconnectivity. The underlying topology considered is a fully connected network of K nodes that exchange messages through a set of wireless communication links. We leave the generalization to other topologies for future work, but the system considered consists of a set of good nodes and a set of faulty nodes. A good node is assumed to be an active participant and correctly execute the algorithms. A faulty node is either benign (detectably bad), symmetric, or bounded-arbitrary (Byzantine) faulty.

A fully connected graph of K nodes consists of $K(K-1)$ unidirectional links. A good link between any two nodes is assumed to correctly deliver a message from its source node to its destination node within a bounded communication delay time. A faulty link does not deliver the message, delivers a corrupted message, or delivers a message outside the expected communication delay time. The communication means is wireless broadcast, i.e., one-to-many, with each node broadcasting on a separate and dedicated channel.

Broadcast of a message by a node is realized by transmitting the message, at the same time, to all nodes. 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]. We assume $F < K/3$ and define the minimum number of good nodes in the system,

G, by $G > 2K/3$ nodes. For node-fault model the minimum number of nodes needed to maintain synchrony is well established to be $3F+1$ [Lam 1982, 1985][Dol 1984].

3.1. Communication Delay

The communication delay between the nodes is expressed in terms of the minimum event-response delay, D, and network imprecision, d. These parameters are measured at the network level. A message broadcast by a node at real time t is expected to arrive at all other nodes, be processed, and subsequent messages generated by those nodes within the time interval $[t+D, t+D+d]$. 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 many factors including, but not limited to, the drift of the oscillators with respect to real time, hence, variation is processing time, jitter, discretization error, temperature effects and especially differences in the distances between the nodes, i.e., the lengths of the links, which is more a prevalent factor in wireless communication. These parameters are assumed to be bounded, $D > 0$, $d \geq 0$, and both have units of real-time clock ticks and their values known in the network. The communication delay, denoted γ, is expressed in terms of D and d, is defined as $\gamma = D+d$, and has units of real-time clock ticks. In other words, we assume synchronous communication and bound the communication delay between any two nodes by $[D, \gamma]$.

3.2. The *Sync* Message And Its Validity

In order to achieve and maintain desired synchrony, the nodes communicate by exchanging *Sync* messages, where *synchrony* is defined as a measure of the relative imprecision of the values of the local clocks of the good nodes, which is algorithm dependent and typically a multiples of γ. In a following section, a formal definition of synchrony is provided for the algorithm used in this paper. A *Sync* message from a given source is *valid* if it arrives at or after one D of an immediately preceding *Sync* message from that source, that is, the message validity in the value domain, i.e., valid *Sync* messages are rate-constrained. Assuming physical-layer error detection is dealt with separately, the reception of a *Sync* message is indicative of its validity in the value and time domains. A node assumes own message to be valid locally γ clock ticks since it was broadcast.

4. The Primary Clock Synchronization Protocol

There exist a family of mechanically verified clock synchronization protocols that address different types of faults [Mal 2011, 2015]. In [Mal 2015] a strategy was proposed to tolerate Byzantine faults indirectly by first converting a Byzantine fault to a symmetric fault, which the *3ROM* algorithm [Mal 2017] does in three communication rounds, and then using a symmetric-fault-tolerant protocol to synchronize the system of K nodes, where $F < K/3$. A symmetric-fault-tolerant protocol was also introduced in [Mal 2015], that we refer to as the Primary protocol, and reproduce it in this section for references but refer the reader to [Mal 2015] for a more in depth analysis and the details of its mechanical verification.

If the types of faults to be tolerated are assumed to be detectable type, the Digraph protocol presented in [Mal 2011] is more suitable since it is more versatile and readily applies to any static and/or dynamic topology. But since detectable faults are a subset of symmetric faults, the symmetric-fault-tolerant protocol presented below is also capable of addressing detectable faults. Thus far, mechanical verification of this protocol has been limited to fully connected graphs. We leave the generalization to other topologies for future works.

Protocol Assumptions

1. The topology is a fully connected graph
2. F is the maximum number of symmetric faults in the network
3. The number of nodes constituting the network is K, where, for node-fault model, $F < K/3$ nodes
4. The bound on the oscillator drift rate is ρ, where $0 \leq \rho << 1$
5. A message sent by a node will be received and processed by the destination nodes within γ, where $\gamma = (D + d)$
6. Physical-layer error detection is dealt with separately, thus, the reception of a *Sync* message is indicative of its validity in value and time domains

4.1. 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 that are directly connected to it, with one monitor for each source of incoming message. A monitor keeps track of the activities of its corresponding source node. A valid *Sync* message is conveyed to the local synchronizer. The assessment results of the monitored nodes are then utilized by the synchronizer in the synchronization process. A monitor disposes of the valid message after its life-span expires.

ValidateMessage():

if (*incoming message* = *Sync*) and (*MessageTimer* ≥ *D*)
    MessageValid = true, *// store it,*
    MessageTimer = 0,

elseif (*MessageTimer* ≥ *MessageLifeSpan*)
    MessageValid = false, *// it expired*

elseif (*MessageTimer* < *MessageLifeSpan*)
    MessageTimer = *MessageTimer* + 1.

Accept():

if (*number of stored Sync messages* ≥ T_A)
    return true,
else
    return false.

Figure 1. The protocol functions.

The function *ValidateMessage()*, shown in Figure 1, is used by the monitors to determine whether a received *Sync* message meets the minimum timing requirement, and thus is valid in both value and time domains, and whether a stored valid *Sync* message has reached its lifespan and expired. The function *Accept()*, used by the synchronizer, examines availability of sufficient valid *Sync* messages. The sufficiency of available, valid messages, denoted by T_A, is a function of the type and number of faults tolerated. For tolerating F_S simultaneous symmetric faults, $T_A \geq K/2$. When a sufficient number of messages have been received, the *Accept()* function returns a Boolean value of true.

4.2. The Symmetric-Fault-Tolerant Clock Synchronization Protocol

Due to inherent drift in local oscillators in the nodes, their clocks are bound to drift apart resulting in gradual degradation of synchrony, as a result, the nodes have to periodically resynchronize, referred to as the *resynchronization process*. The resynchronization process begins when the first good node times out and transmits a *Sync* message and ends after an accept event (as defined in the protocol) occurs in every good node. Upon start of a new round of a resynchronization process, a node continually sends out *Sync* messages, once per γ, to other nodes that are connected to it. Consequently, the life-span of a *Sync* message at the receiving nodes is set to be γ. The protocol, executed by all good nodes, is presented in Figure 2 and consists of a synchronizer and a set of monitors that execute once every local clock tick. Four concurrent *if* statements collectively describe the synchronizer. These statements are labeled ST (*StateTimer*), LT (*LocalTimer*), TS (Transmit *Sync*), and TT (*TransmitTimer*). The function *ValidateMessage()* describes the monitor.

<table>
<tr>
<td><u>Synchronizer:</u>
ST1: if (StateTimer < 0) or (Accept())
  StateTimer := 0,   // reset
ST2: elseif (StateTimer < P_{ST})
  StateTimer := StateTimer + 1.</td>
<td rowspan="2">TT1: if (TransmitTimer < 0) or
  ((TransmitTimer ≥ γ) and
  (StateTimer ≥ P_{ST}))
  TransmitTimer := 0,
TT1: elseif (TransmitTimer < γ)
  TransmitTimer := TransmitTimer + 1.</td>
</tr>
<tr>
<td>LT1: if (LocalTimer < 0) or
  (LocalTimer ≥ P_{LT}) or
  (StateTimer = $\lceil \pi_{init} \rceil$)
  LocalTimer := 0,   // reset
LT2: else
  LocalTimer := LocalTimer + 1.</td>
<td>TS1: if (StateTimer ≥ P_{ST}) and // timed out
  (TransmitTimer ≥ γ) and
  (not Accept())
  Transmit Sync.</td>
</tr>
<tr>
<td colspan="2"><u>Monitor:</u>
  ValidateMessage().</td>
</tr>
</table>

Figure 2. The symmetric-fault-tolerant protocol.

The following symbols are used in Figure 2 and in stating the following self-stabilization properties:

- P_{LT} 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, $P_{LT} >> 0$.
- P_{ST} 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 *StateTimer* by a node, $P_{ST} >> 0$.
- $\Delta_{Net}(t)$, 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 $t \geq C$, $0 \leq \pi << P_{LT}$.
- C, the convergence time, is defined as the bound on the maximum time for the network to achieve the guaranteed precision π.

A distributed system is defined to be self-stabilizing if, from an arbitrary initial state, it is guaranteed to reach a legitimate state in a finite amount of time and remain in a legitimate state. For clock synchronization, a *legitimate state* is a state where all parts in the system are in synchrony which is formally defined below [Mal 2011]. To prove that a protocol is self-stabilizing (self-synchronizing), it suffices to show that the following self-stabilization properties hold.

1. **Convergence:** $\Delta_{Net}(C) \leq \pi$, $0 \leq \pi << P_{LT}$
2. **Closure:** For all $t \geq C$, $\Delta_{Net}(t) \leq \pi$
3. **Congruence:** For all nodes N_i, for all $t \geq C$, $(N_i.LocalTimer(t) \leq \lceil \pi \rceil)$ ➔ $\Delta_{Net}(t) \leq \pi$.
4. **Liveness:** For all $t \geq C$, good node N_i, $i = 1..K$, there exists $(P_{ST} - \pi - \gamma) \leq U \leq P_{ST}$, such that $N_i.LocalTimer(t+1) = \text{mod}(N_i.LocalTimer(t)+1, U)$.

In the context of this paper, we set $C = P_{LT} + ResetLocalTimerAt + 2\gamma$, where *ResetLocalTimerAt* is a time when the *LocalTimer* is reset and we chose $\lceil \pi_{init} \rceil$ as the earliest time when all good nodes have completed the resynchronization process. Since $0 < \gamma << P_{ST} < P_{LT}$, and the *LocalTimer* is reset after reaching P_{LT} (worst-case wraparound), a trivial solution is not possible.

The presented algorithm provides a guaranteed initial precision of $\pi_{init} = d + \gamma + \delta(d+\gamma) < 2\gamma$ clock ticks [Mal 2015], which is referred to as coarse synchrony. Hereafter, we refer to this protocol as the Primary clock synchronization algorithm, or simply the Primary algorithm. In the next section, we introduce a secondary clock synchronization protocol that achieves the theoretical precision of one clock tick.

4.3. What Self-Stabilization Properties Mean

The *Convergence* and *Closure* properties address achieving and maintaining network synchrony, respectively. As formally defined in the previous section, given sufficient time, C, the convergence property examines whether or not the system has reached a point where all nodes are within the specified precision. The closure property, on the other hand, examines whether or not the system starting within the specified precision will remain within that precision thereafter.

The convergence and closure properties provide an external view of the system, whereby the external viewer can examine whether or not the system has self-stabilized.

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. More precisely, TDMA-type applications are based on the fundamental assumption of the existence of initial synchrony. The protocol presented in this report is meant to provide this fundamental requirement of TDMA-type applications to higher-level protocols. One of the challenges in employing multiple protocols in distributed system has been the integration of these protocols operating at different levels of application. Previously, the integration of a lower-level protocol with higher-levels either has not been addressed or had simply been overlooked. The *Congruence* property, therefore, addresses this essential requirement. Unlike the convergence and closure properties that provide system view from the perspective of an external viewer, the Congruence property provides a local view from the perspective of a node by providing the necessary and sufficient conditions for the node to locally determine whether or not the system has converged. The Congruence property, therefore, is essential in the integration of this underlying self-stabilization protocol with higher-level protocols in the system.

The *Liveness* property examines whether or not a node takes on all possible discrete values within an expected range. In other words, the system is "alive" and the good nodes execute the protocol properly, and time advances within each node.

5. The Secondary Clock Synchronization Protocol

The protocol presented in this section achieves fine synchrony, i.e., the theoretical precision of one clock tick, across a distributed network of K nodes. We do not assume prior knowledge of the location of each node or the distances between any two nodes; however, we assume the following.

Protocol Assumptions

1. The topology is a fully connected graph.
2. F is the maximum number of asymmetric (Byzantine) faults in the network
3. The number of nodes constituting the network is K, where, for node-fault model, $F < K/3$ nodes
4. The bound on the oscillator drift rate is ρ, where $0 \leq \rho << 1$
5. Known maximum communication delay between any two nodes, $\gamma \geq (D + d)$
6. Coarse initial synchrony with $\pi_{init} \leq 2\gamma$ clock ticks
7. Unique identifier (ID) for each node and nodes are ordered counter clockwise, looking down from high above and assuming nodes not to be in the same plane as the reference point
8. Each node broadcasts its messages on a separate channel
9. Physical-layer error detection is dealt with separately, thus, the reception of a Sync message is indicative of its validity in value and time domains

The Secondary Clock Synchronization Protocol

The protocol below is executed by all nodes N_i, for all i, every clock tick. This protocol is based on the assumption of initial coarse synchrony ($\pi_{init} << P_{LT}$). To maintain consistency with terminologies used in the literature, we use the terms *Init* and *Echo* for messages as in [Sri 1987]; however, in our protocol, the *Echo* message is a vector of locally time-stamped events. We should add that there are other clock synchronization algorithms that are based on the assumption of initial synchrony, e.g., [Pea 1980], [Wel 1988], and the broadcast primitive algorithm in [Sri 1987]. The three referenced algorithms, [Sri 1987][Pea 1980][Wel 1988], achieve optimum synchrony in multiple rounds of iterations while our solution achieves optimum synchrony in only one iteration. Furthermore, although not explicitly stated, these algorithms assume a fully connected graph with traditional node-fault model and are not necessarily tailored for wireless communications. Our proposed solution, however, is specifically designed with wireless communications in mind. Since we accommodate for larger variation in the communication latencies among the nodes than the algorithms tailored for wired networks, our solution equally applies to both wireless and wired networks.

A Fault-Tolerant Clock Synchronization Protocol For Wireless Networks, Figure 3, hereafter referred to as the Secondary algorithm, starts executing when prompted by the Primary algorithm, via the Congruence property, that the network is in synchrony, albeit, coarse synchrony. Assertion of the Congruence property also resets local state of the node in preparation for the Secondary algorithm. Conversely, execution of the Secondary algorithm is halted when the Congruence property no longer holds.

- $\omega = \pi_{init} + \gamma$
- $\psi = ResetLocalTimerAt$
- *Init*, a message broadcast by a node to all others.
- *Echo*, a message broadcast by a node to all others and is a vector of K entries of time-stamped events indicative of arrival times of the *Init* messages. A node assumes its own messages (*Init* and *Echo*) to be valid γ clock ticks after it broadcasts them. We assume similar validity measures (given the expected time intervals) for the *Init* and *Echo* messages as we did for the *Sync* message.

```
BI:     if (LocalTimer = ψ)
            Broadcast Init
BE:     if (LocalTimer = ω + ψ)
            Broadcast Echo
RCA:    if (LocalTimer = 2ω + ψ)
            Recover(), Adjust()
```

Figure 3. A Fault-Tolerant Clock Synchronization Protocol For Wireless Networks.

We now describe the *Recovery()* and *Adjust()* functions used by the protocol. Let, at any node N_x, M be the matrix of received messages, where a row i is a vector of locally time-stamped values received from node N_i (content of received *Echo* message from N_i). Hence, a column j is

the vector of reportedly received values from N_j. Thus, *M(i, j)* is the time when N_i is reported to have received a message from N_j. Let *T* be a matrix of time-differences between nodes N_i and N_j.

$$T(i,j) = (M(i,j) - M(j,i)) / 2 \quad (1)$$

Thus, matrices *M* and *T* have same dimensions. In evaluating Equation 1, *T(i,j)* is invalid if the right hand side of the equation contains an invalid value. It follows from Equation 1 that *T* is skew-symmetric and an invalid entry in *M*, ex. *M(i,j)*, will result in two invalid entries in *T*, *T(i,j)* and *T(j,i)*.

The distance between any two nodes N_i and N_j is the average of the time of received messages exchanged between the two nodes multiplied by the speed of light, *C*, and is determined by the following equation. The distance is unknown if the right-hand side contains an invalid value.

$$D_{ij} = C (M(i,j) + M(j,i)) / 2 \quad (2)$$

Recovery()

In this section we describe the *Recovery()* function that, in turn, consists of two parts, recovering invalid *Init* and recovering invalid, or missing, *Echo* messages.

As we discussed earlier in this paper, synchrony is a prerequisite in using trilateration for determining the location and, consequently, distances to the intended object. Matrix *T* is introduced to aid with the recovery of missing data, provide fault-tolerance, and achieving fine synchrony given initial contents of matrix *M*. Once faults are recovered, fine synchrony is achieved and the contents of matrix *M* is restored.

Recovering Invalid Init

Recall that a fault is defined as no message or an invalid received message. A faulty/no *Init* message manifests as an invalid entry in matrix *M*. As long as the fault assumptions are not violated, recovery of an invalid *Init* is possible by using valid data received by other nodes. In particular, a fault between nodes N_i and N_j can be recovered as long as there is valid data between these nodes and a third node N_x.

$$T(i,j) = T(i,x) - T(x,j) \quad (3)$$

Note that a missing entry in *M(i,j)* is synonymous to a link fault. Having *T(i,j)* and either *M(i,j)* or *M(j,i)*, missing either *M(j,i)* or *M(i,j)* is reconstructed by using Equation 1. An example is provided in Appendix B. But, as we mentioned earlier, after the network has reached fine synchrony, D_{if} is determined using trilateration and data in *M*, provided there is sufficient data in *M*. Using Equation 4, derived from Equations 1 and 2, $M(i,_f)$, and subsequently $M(_f,i)$, are recovered. In case of marginally sufficient data, where two possible solutions exist, the assumption of an ordered network lends itself to determining the correct solution.

$$M(i,j) = T(i,j) + D_{ij} \quad (4)$$

We would like to point out that a missing entry in *M(i,j)* is synonymous to a link fault. Also, it follows from Equation 3 that if a node is silent and does not broadcast *Init* and *Echo* messages, it cannot be recovered. On the other hand, if a node broadcasts *Init* message to at least three good nodes but does not broadcast an *Echo* message, it can be recovered as described next. We leave diagnosis of the network and analysis under various fault scenarios to future work.

Recovering Invalid Echo

A faulty *Echo* message manifests as an invalid row in matrix M. Let, at N_i, N_f be the faulty node whose corresponding row in M contains no data. Let V be a vector of data associated with all nodes that received valid *Init* messages from N_f, i.e., V is in the column f in M and $V = M(i,f) = valid$. The following iterative algorithm recovers from this fault and restores invalid row f of M, provided the fault assumptions are not violated, i.e., there exist sufficient valid entries in V. The number of iteration is captured by w.

1. Determine D_{ij} using Equation 2, for all $i, j, N_i \neq N_f$ and $N_j \neq N_f$.
   Reset iteration counter w.
2. Realign all nodes in V, around node N_j from the set of nodes whose values constitute the vector V, excluding N_f, by adjusting the content of the vector V as described in the Equation 5. Although typically a node uses itself as reference, an optimum reference for alignment would be choosing the node whose value is at the midpoint of the values in V.
   $$V(i) = M(i, f) + T(j,i), \text{ for all } i \quad (5)$$
3. Use trilateration with entries in V – modulo the faulty node's – and the location of those nodes relative to each other, i.e., D_{ij}, to determine the time when N_f had broadcast its message. Repeat this process until one of the following conditions *a* or *b* is satisfied, otherwise, adjust V by some amount $0 < x < \gamma$ and continue, where x is a fraction/multiple of clock ticks.
   $V(j) = V(j) - x$, for all j
   Increment iteration counter w
   a. Trilateration (using the values in V) results in a closest intersecting point, where any two intersecting points are within $\delta \geq 0$ of each other, and so a solution exists. The amount of imprecision, $0 \leq \delta << \gamma$, is due to drift and noise.
   b. Trilateration does not converge to a closest intersecting point after $w \geq \pi_{init}/x$ iterations and so there does not exist a solution.

4. If there exists a solution, the intersecting point is indicative of the time when N_f had broadcast its *Echo* message and xw is the amount of time it took to reach the convergence point. Reconstruct $T(i,f)$ as follows.
   $T(j,f) = xw$, where N_j is the reference node used in Step 2
   $T(i,f) = T(j,f) - T(j,i)$, for all i and $i \neq j$
   $T(f,i) = -T(i,f)$, to preserve symmetry in T

   Repair M using T and Equation 1.
   $M(f,i) = M(i,f) - 2T(i,f)$, for all i

   Find the remaining distances D_{ij} between all nodes using Equation 2.

Having accurately measured the distances between any two nodes, and since the node ID's are assumed to be ordered, the geometry of the network in 3-dimensions is uniquely determined if the projection of the nodes onto the x-plane (ground) maintains their ID order.

Adjust()

The purpose of this function is to adjust the local time of the nodes to a reference point in time and, thus, establish an optimum synchrony across the network. Construct a timeline of transmission times of *Init* messages of all nodes using a given row of the matrix *T* (typically a node uses own row). Although the reference point in time can be anywhere on the timeline, to tolerate *F* faults, given the assumptions hold, we discard *F* values from both extremes and choose the midpoint of the two remaining extremes values (transmission times). The process of choosing the reference point has to be consistent at all nodes. Let *LT* and *RT* be the left and right most transmission times of the remaining nodes on the timeline, respectively.

$$t_{MidPoint} = (RT + LT) / 2$$

The adjustment amount is determined by the following equation that is then incorporated into the node's local timer.

$$Adj = (RT + LT) / 2 = t_{MidPoint}$$
$$LocalTimer = LocalTimer - Adj$$

Hereafter, by synchronized network we mean the network precision is within the theoretical bound of one clock tick.

6. Locating An Object

Assuming a network of synchronized nodes, an object is accurately located based on its capabilities and the extent of its interactions with the nodes. We enumerate the following scenarios. Figure 4 is a depiction of typical section of a farm for crop-dusting application.

1. **Active Participant** - If the object has similar capabilities to the nodes and participates in the synchronization process, it determines its own location from the exchanged messages, Figure 4(a).
2. **Passive Participant** - If the object has similar but fewer capabilities to the nodes, where it receives all exchanged messages between the nodes but does not participate in the synchronization process, i.e., it is a passive participant, it determines its own location from the exchanged messages between the nodes, Figure 4(b).
3. **DLPS** - Analogous to GPS satellites, the nodes periodically broadcast their locations and times, $((x, y, z), t)$, and the object determines its location from the received data, Figure 4.
4. **DLPS Radar** - The object periodically broadcasts a signal or emits a signature whereby it is identified by the nodes, its location is accurately determined by the nodes (as described in the following section), and the nodes broadcast the object's location, $((x, y, z), t)$, back to the object. The object determines its own location with a simple vote (or any other method of combining data), which is a drastic reduction in on-board processing requirement of the received data from multiple nodes, Figure 4.

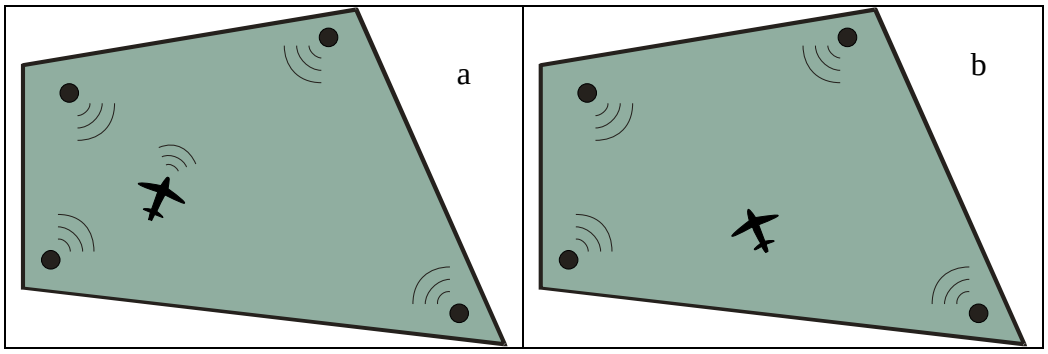

Figure 4. Active and passive participants.

In practice, options 3 and 4 can be implemented even if the object's capability is limited to one receiver; the nodes broadcast their data using a TDMA schedule on a single communication channel, thus, preventing interference when communicating with the object.

6.1. PAssive DIstributed Detection And Ranging System (PADIDAR)

The Oxford Dictionary defines a radar as "*a system for detecting the presence, direction, distance, and speed of aircraft, ships, and other objects, by sending out pulses of high-frequency electromagnetic waves that are reflected off the object back to the source.*" We call such a system an *active radar*. In contrast, similar functionality can be achieved by a set of sensors at the nodes that are tailored to the object's signature or a signal broadcast by the object. We call such a system a *passive radar*.

Indeed, given a set of connected and synchronized nodes with appropriate sensors, and given an object periodically broadcasting a signal or emitting a signature (whereby it is identified by the nodes), the nodes can detect the presence, distance, location, and, subsequently, determine the trajectory of the object properly and accurately and, thus, constitutes a *PAssive DIstributed Detection And Ranging* system (PADIDAR). In this context, PADIDAR is a one-way ranging system. We define PADIDAR as $R = f(Sensor, Object)$. The following table lists some examples of the Sensor-Object pairs.

Table 1. Examples of Sensors and Objects.

Sensor	**Object**	**Intent**
Sound/Noise	Bullet, Missile, UAV, Ball (tennis, football, etc.)	Location, Projection
Vision/Light/Heat	UAV, Missile, Meteor	Location, Projection
Wave/Vibration	Water, Light	Locating earthquake epicenter

We consider the following two scenarios for different behaviors of the object.

Scenario A: The object actively and periodically broadcasts a signal, Figure 4(a) – Since an object's data arrives out of synchrony with the synchronized nodes, the nodes need to first determine when the object had broadcast its signal in order to locate its position. Thus, when a node receives/detects a signal from the object, it immediately broadcasts this locally time-stamped event to other nodes. Since the nodes are synchronized with each other, each node will

receive the object data from all other nodes within one γ. As a result, initial precision (time lag) of the object data among the nodes is now γ. The exchanged object data is stored locally in a vector V at a node (similar to the previous section).

To accurately locate the object, the nodes execute the following algorithm that is a shorter version (Steps 3 and 4) of the algorithm presented in a previous section entitled "***Recovering Invalid Echo***", keeping in mind that D_{ij} is the distance between nodes i and j, and N_f is now referred to the object.

1. If received/detected signal from object, broadcast it.
2. Use trilateration with entries in V and D_{ij} to determine the time when the object had broadcast its message. Repeat this process until one of the following conditions *a* or *b* is satisfied, otherwise, adjust V by some amount $0 < x < \gamma$ and continue, where x is a fraction/multiple of clock ticks.
   $V(i) = V(i) - x$, for all i
   Increment the iteration counter w

   a. They converge to a closest intersecting point so that the difference between any two values in V is within δ, $0 \leq \delta << \gamma$, and so a solution exists.
   b. They do not converge after $w \geq \gamma/x$ iterations and so there does not exist a solution.

3. If there exists a solution, the intersecting point is the time when the object had broadcast its message and xw is the amount of time it took to reach the convergence point and the current content of V holds the broadcast time of the object's message.

Scenario B: The object is passive and does not periodically broadcast a signal, Figure 4(b) – The object is assumed to continuously emit a signature, e.g., sound/light/heat emanating from a meteor or the object's presence is detected using visual sensors/cameras at the nodes. In this scenario, the synchronized nodes sample the object's signature at specific intervals. To simplify the process, the nodes sample and exchange object data with each other along with their synchronization message exchange activities, therefore, eliminating the need for timing alignment that was necessary for *Scenario A*. The exchanged object data is stored at a node in a vector V, similar to Scenario A.

At the end of either *Scenario A* or *B*, the proper timing of the object data is reflected in V and, thus, the distances between the nodes and the object are readily computed and the location of the object is determined using trilateration. Note that since the timing values in V are optimum, this final localization step does not require further iterations.

6.2. Dynamic Geometry And Mobility

The reliability and accuracy of such a positioning system fundamentally depends on two factors; the accuracy of its time-synchrony, at the node and network level, and its geometry, i.e., locations and distances of its nodes from each other. Just as geo-location using GPS data is based on the knowledge of the locations and current times of the satellites, current ground-based geo-location solutions are also based on prior knowledge of the exact location of the beacons

(nodes) and the distances between them; the more accurate the data (time and distances), the more accurate the location of the object is estimated. For instance, Locata [Riz 2013] ground-based LPS, requires exact knowledge of the location of its beacons and their distances; thus, a survey of the locations of the beacons and their geometry has to be conducted and the survey results shared by all beacons prior to its intended operation. The requirement of prior knowledge of network geometry imposes a restriction on the applications and precludes scenarios where the nodes are mobile.

In contrast, the self-synchronization protocols presented in this paper are autonomous and not only establish an accurate time-synchrony, they also determine the network geometry (the relative locations and distances of the nodes with respect to each other) without requiring such prior knowledge or the help of an external source. The autonomous distributed fault-tolerant local positioning system presented, therefore, provides added capability for high-dynamic environments, where not only the object is assumed to be mobile and maneuver at high speed, but the nodes are also assumed to be static and/or mobile and their locations, potentially, change as a function of time. The only restriction being that the nodes and the object remain within the maximum communication range of γ of each other. Figure 5 depicts a 2D (the concept is equally applicable to 3D) example of a fully mobile system, where the locations of the nodes and the object change as a function of time.

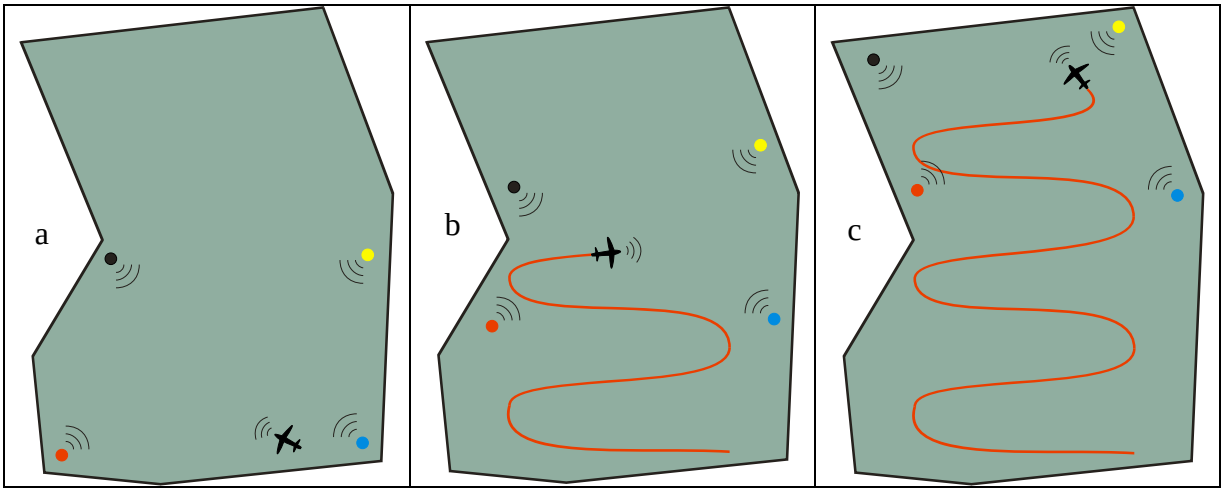

Figure 5. Dynamic geometry and mobility.

7. Integrated Self-Synchronizing Algorithm

Together, the Primary and Secondary algorithms presented in previous sections guarantee bringing a system to a safe (legitimate) state, even when started from an arbitrary initial configuration, and achieve network level fine synchrony. These algorithms are the building blocks of the autonomous distributed fault-tolerant local positioning system.

The accuracy of the localization provided by this local positioning system is a function of the frequency of data it broadcasts, e.g., ((x, y, z), t), and the measure of synchrony among its

distributed components, which, in turn, is a function of the drift of their local oscillators. Thus, to provide data with the highest accuracy, we must broadcast localization data as frequently as possible and maintain synchrony among the nodes as often as practicable. Note that performances of the synchronization and localization algorithms are bounded by the communication latency γ, thus, imposing a limit on broadcast frequency.

Earlier in this report we had also pointed out that, due to drift in local oscillators, undergoing the periodic resynchronization process is a necessary part of these algorithms. The Congruence property indicates the Primary algorithm's successful completion of the resynchronization process, and triggers start of a new round of resynchronization process by the Secondary algorithm. Executing these algorithms in sequence, however, introduces undesirable hiccups in the system. As discussed in [Mal 2014], in the Primary algorithm, the *LocalTimer* is used to provide a jitter-free clock to the higher level protocols by properly filtering out inherent deviation (hiccup) in the *StateTimer* during the resynchronization process. During this process accuracy of the local positioning system data is less than desired. In addition, as the clocks drift the accuracy of the geo-location data diminishes. Thus, the more the rate of drift of the oscillators, the more frequent the resynchronization process needs to take place; however, if the nodes' local timers maintain a high degree of relative synchrony, we may not need to undergo the periodic resynchronization process by the Primary algorithm. We would like to point out that the need for the resynchronization process cannot be fully eliminated and will be reserved for circumstances when the system experiences unexpected loss of synchrony that will be indicated by the Congruence property.

When the nodes are synchronized, *BI* and *BE* phases of the Secondary algorithm each take one γ to complete while *RCA* phase takes negligible time to compute compared to the communication latency γ, Figure 3. In other words, each iteration of the Secondary algorithm takes 2γ clock ticks. Proper overlapping of the *BI* and *BE* phases halves the time interval of iteration of the Secondary algorithm, reduces the amount of drift among the local timers of the nodes, thus, maintaining higher synchrony in the system, provides higher frequency and more accurate localization data. In this section we introduce an algorithm that is an amalgamation of the Primary and Secondary algorithms while eliminating inefficiencies and, at the same time, speeding up the resynchronization process.

Since broadcast frequency is bounded by γ, for geo-location purposes, localization data is broadcast periodically at γ intervals; however, when augmented by adding an *Echo* message, it also facilitates maintaining a higher level of relative synchrony in the system. The benefits of this approach far outweighs the extra overhead of adding *Echo* since it eliminates the need to undergo the periodic resynchronization process and the resulting hiccups in the system.

The integration of the two algorithms is subtle and based on the idea of augmenting the geo-location data, that includes nodes' local times, $((x, y, z), t)$, for clock synchronization purposes. To properly integrate the two algorithms, the statements *ST1* and *TS1* of the Primary algorithm need to be modified in order to prevent unnecessary periodic resynchronization after achieving synchrony, Figure 2 and Figure 6.

Synchronizer:

ST1: if (*StateTimer* < 0) or (*Accept()*) or (*StateTimer* ≥ P_{ST} and *InSynch*)
StateTimer := 0, // *reset*

ST2: elseif (not *InSynch*)
if (*StateTimer* < P_{ST})
StateTimer := *StateTimer* + 1
else

TS1: if (*StateTimer* ≥ P_{ST}) and // *timed out*
(*TransmitTimer* ≥ γ) and
(not *Accept()*) and (not *InSynch*)
Transmit *Sync*.

Figure 6. Modification to symmetric-fault-tolerant protocol.

The integrated self-synchronization algorithm is presented in Figure 7. Start of this algorithm is triggered by the Congruence property when the Primary algorithm achieves coarse synchrony.

SnotT: *While ((not InTransition) and (InSynch))*
if ((LocalTimer mod γ*) = 0)*
Recover(), Adjust(), UpdateEcho()
Broadcast Echo

TnotS: *if (InTransition and (not InSynch))*
if (LocalTimer = ψ*)*
Broadcast Init
if (LocalTimer = $\omega + \psi$*)*
Broadcast Echo
if (LocalTimer = $2\omega + \psi$*)*
Recover(), Adjust(), UpdateEcho()
if (LocalTimer = 3ω*)*
InSynch = true
InTransition = false
Broadcast Echo

Figure 7. The fault-tolerant integrated self-synchronizing algorithm.

- $\omega = \pi_{init} + \gamma$, and $\psi = ResetLocalTimerAt$
- *InTransition*
  - Set by the Primary algorithm, Congruence property
  - Reset by the Primary algorithm when synchrony is lost, Congruence property, or by the Secondary algorithm upon completion of the transitory interval, statement *TnotS*.
- *InSynch*
  - Set by the Secondary algorithm upon completion of the transitory interval
  - Reset by the Primary algorithm when synchrony is lost, Congruence property

The *Recovery()* and *Adjust()* functions were described earlier. We describe *UpdateEhco()* function here. But, since the *Echo* message is broadcast at γ intervals (after achieving network-level synchrony), the *Recovery()* function is revisited. An example of execution of this algorithm is provided in the Appendix A.

Recovering Invalid Echo Revisited

A faulty *Echo* message manifests as an invalid row in matrix *M*. Let, at N_i, N_f be the faulty node whose corresponding row in *M* contains no data. Let *V* be a vector of data associated with all nodes that received valid *Init* messages from N_f, i.e., *V* is the column *f* in *M* and $V = M(i,f) = valid$. For error detection and recovery purposes, we build a new matrix, *M-New*, for the newly arrived messages. If N_f had broadcast in previous round (equivalent to *Init* messages) to sufficient number of good nodes (three/four) but did not broadcast *Echo*, N_f is recovered using trilateration. The matrix *M* is restored using the algorithm described in a previous section entitled "*Recovering Invalid Echo.*" With the integrated algorithm of Figure 7, if the assumptions hold, a node exhibiting faults at every other interval is readily tolerated.

If the nodes are not mobile and the assumptions hold, a simpler (shortcut) solution to recovery is to copy the N_f column/row in the *M-New* matrix using timing values from its *Init* message in the *M* matrix.

$M\text{-}New(i,N_f) = M(i,N_f)$

and

$M\text{-}New(N_f,j) = M(N_f,j)$, for all *i* and *j*.

UpdateEcho()

Construct the *Echo* vector, as the messages arrive, in preparation for the next round.
For all sources of a messages *i*,

```
if (InSynch)
    Echo(i) = LocalTimer mod γ
    if Echo(i) = 0
        Echo(i) = γ
```

An implication of the above modulus operation is that P_{LT} needs to be a multiples of γ to avoid complications arising due to fractions of γ that would result otherwise.

8. Conclusions

We have described an autonomous distributed fault-tolerant local positioning system, a fault-tolerant GPS-independent distributed autonomous distributed local positioning system, for static and/or mobile objects and presented solutions for providing highly-accurate geo-location data for the static and/or mobile objects in dynamic environments. We have explored the fundamental issues governing a distributed local positioning system, namely, timeliness of its broadcasting signals and the knowledge of its geometry, i.e., locations and distances of the beacons. We also addressed shortcomings of existing distributed positioning systems, namely, a single point of failure and lack of addressing various fault manifestations, in particular, communication link failures. Our proposed solution, solves this problem by first employing fault-tolerant distributed clock synchronization protocols to achieve the theoretical synchrony of one-clock tick across the distributed system of nodes (beacons) and, consequently, determine the geometry of the network, and then use trilateration to accurately determine the current location of the intended object. The presented solution is an algorithm that is an amalgamation of the Primary and Secondary algorithms while eliminating inefficiencies and, at the same time, speeding up the resynchronization process. We presented a new synchronization protocol entitled "*A Fault-Tolerant Clock Synchronization Protocol For Wireless Networks*," which we referred to as the Secondary algorithm. We also addressed subtleties of integrating the two algorithms, which are based on the idea of augmenting the geo-location data, which includes nodes' local times, $((x, y, z), t)$, for clock synchronization purposes. Although we addressed fault detection to some extent, we leave full investigation of this issue to future work. We would like to point out that comparisons of various matrices at proper timing events can help with the detection of various faults. Also, due to time constraints we restricted our topology to fully connected graphs and leave generalization of this work to future work. Mechanical verification and formal proof of the proposed solution are left to future work.

References

[Bie 2011] Biely, M.; Schmid, U.; Weiss, B.: *Synchronous consensus under hybrid process and link failures*, Journal of Theoretical Computer Science, vol. 412, no. 40, pp. 5602-5630, 2011.

[Con 2006] Constellation CxP 70024 (BASELINE), Constellation Program Human-Systems Integration Requirements, Glossary, December 2006.

[Dol 1984] Dolev, D.; Halpern, J.Y.; Strong, R.: *On the Possibility and Impossibility of Achieving Clock Synchronization*, proceedings of the 16th Annual ACM STOC (Washington D.C., Apr.). ACM, New York, 1984, pp. 504-511. (Also appear in J. Comput. Syst. Sci.)

[Dri 2003] Driscoll, K.; Hall, B.; Sivencrona, H.; Zumsteg, P.:, *Byzantine Fault Tolerance, from Theory to Reality*, LNCS, 22nd International Conference on Computer Safety, Reliability and Security, pp. 235-248, September 2003.

[Hay 2014] Hayhurst, K.J.; Maddalon, J.M.; Morris, A.T.; Neogi, N.; Verstynen, H.A.: *A Review of Current and Prospective Factors for Classification of Civil Unmanned Aircraft Systems*, NASA/TM-2014-218511, August 2014.

[Hay 2015] Hayhurst, K.J.; Maddalon, J.M.; Neogi, N.; Verstynen, H.A.: *A case study for assured containment*, 2015 International Conference on Unmanned Aircraft Systems (ICUAS), pp. 260-269, June 2015.

[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.

[Mal 2011] Malekpour, M.R.: *A Self-Stabilizing Synchronization Protocol For Arbitrary Digraphs*, The 17th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC 2011), pp. 10, December 2011.

[Mal 2015] Malekpour, M.R.: *A Self-Stabilizing Hybrid-Fault Tolerant Synchronization Protocol*, 2015 IEEE Aerospace Conference, pp. 11, March 2015.

[Mal 2017] Mahyar R. Malekpour: *Achieving Agreement In Three Rounds With Bounded-Byzantine Faults*, AIAA SciTech 2017, pp. 10, January 2017, to appear.

[Min 2004] Miner, P.S.; Geser, A.; Pike, L.; Maddalon, J.: *A Unified Fault-tolerance Protocol*, In Yassine Lakhnech and Sergio Yovine, editors, Formal Techniques, Modeling and Analysis of Timed and Fault-Tolerant Systems, volume 3253, pp. 167-182, Springer, 2004.

[Pea 1980] Pease, M.; Shostak, R.; and Lamport, L.: *Reaching agreement in the presence of faults*, Journal of the ACM, 27(2): 228-234, April 1980.

[Riz 2013] Rizos, C.: *Locata: A Positinging System for Indoor and Outdoor Applications Where GNSS does not Work*, Proceedings of the 18th Association of Public Authority Surveyors Conference, 2013.

[Sch 2002] Schmid, U.; Weiss, B.; Rushby, J.: *Formally Verified Byzantine Agreement in Presence of Link Faults*, in 22nd International Conference on Distributed Computing Systems (ICDCS'02), pp. 608–616, July 2002.

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

[Wel 1988] Welch, J.L.; Lynch, N.: *A New Fault-Tolerant Algorithm for Clock Synchronization*, Information and Computation volume 77, no. 1, pp.1-36, April 1988.

[GPS] http://www.gps.gov/

Appendix A

The purpose of this example is to give the reader a quick review of and help in understanding the behavior of the integration algorithm. The following is an example of a fully connected graph consisting of 4 nodes, where $F = 0$. Table A.1 shows an execution trace of the network and has six columns; one for time reference, one for each good node listing value of its *LocalTimer*, and the last column is for network precision, π. Each row depicts activities of all good nodes at the corresponding time. Cell contents for the node columns consist of a number corresponding to the value of the *LocalTimer* of the node in conjunction with an activity: 1) *Init* and *Echo* if the node transmits the message, and 2) *Update* if the node corrects its local timer. The received messages at a node are depicted in superscripts, one position for each corresponding node, where a '-' means no messages from that node and an 'i' or 'e' means an *Init* or *Echo* message, respectively, was received.

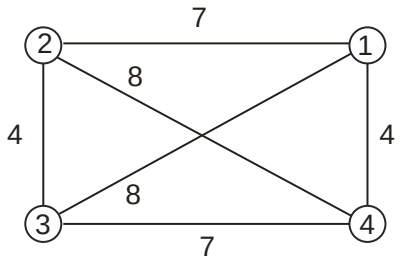

Figure A1. A 4-node network

<u>System parameters:</u>
$D = 4$ clock ticks, $d = 4$ clock tick → $\gamma = 8$ clock ticks
$K = 4$ nodes, $G = 4$ nodes, $F = 0$ nodes
$\psi = ResetLocalTimerAt = \gamma = 8$ clock ticks
$P_{ST} = 1000$ clock ticks
$0 \leq \rho << 1$ → $\delta(P_{ST}) = 0$ clock ticks
$\pi_{init} = d + \gamma + \delta(d + \gamma)$ → $\pi_{init} = 16$ clock ticks (worse case)
$\omega = \pi_{init} + \gamma = 3\gamma = 24$
$\pi = \pi_{init} + 2\delta(P_{ST}) \geq 0$ → $\pi = 16$ clock ticks
$r = \lceil \pi (1 + \rho) \rceil = 16$ clock ticks
$P_{LT} = 1030$ clock ticks

Table A.1. An execution trace of a network of 4 nodes.

Time	$N_{1.LocalTimer}$	$N_{2.LocalTimer}$	$N_{3.LocalTimer}$	$N_{4.LocalTimer}$	π	*InTrs*	*InSyc*
t + 0	8^{----}, *Init*	2^{----}	$?^{----}$	2^{----}	?	T	F
t + 4	12^{----}	6^{----}	$?^{----}$	6^{i---}	?	T	F
t + 6	14^{----}	8^{----}, *Init*	$?^{----}$	8^{i---}, *Init*	?	T	F
t + 7	15^{----}	9^{i---}	$?^{----}$	9^{i---}	?	T	F
t + 8	16^{i---}	10^{i---}	0^{i---}	10^{i---}	16	T	F
t + 10	18^{i--i}	12^{i---}	2^{ii--}	12^{i---}	16	T	F
t + 11	19^{i--i}	13^{i---}	3^{ii--}	13^{i---}	16	T	F
t + 13	21^{ii-i}	15^{i---}	5^{ii-i}	15^{ii--}	16	T	F
t + 14	22^{ii-i}	16^{ii--}	6^{ii-i}	16^{ii-i}	16	T	F
t + 16	24^{ii-i}	18^{ii-i}	8^{ii-i}, *Init*	19^{ii-i}	16	T	F
t + 20	28^{ii-i}	22^{iiii}	12^{ii-i}	22^{ii-i}	16	T	F
t + 23	31^{ii-i}	25^{iiii}	15^{ii-i}	25^{iiii}	16	T	F
t + 24	32^{iiii}, *Echo*	26^{iiii}	16^{iiii}	26^{iiii}	16	T	F
t + 28	36^{iiii}	30^{iiii}	20^{iiii}	30^{eiii}	16	T	F
t + 30	38^{iiii}	32^{iiii}, *Echo*	22^{iiii}	32^{iiii}, *Echo*	16	T	F
t + 31	39^{iiii}	33^{eiii}	23^{iiii}	33^{eiii}	16	T	F
t + 32	40^{eiii}	34^{eiii}	24^{eii}	34^{eiii}	16	T	F
t + 34	42^{eiie}	36^{eiii}	26^{eeii}	36^{eiii}	16	T	F
t + 37	45^{eeie}	39^{eiii}	29^{eeie}	39^{eiii}	16	T	F
t + 38	46^{eeie}	40^{eeie}	30^{eeie}	40^{eeie}	16	T	F
t + 40	48^{eeie}	42^{eeie}	32^{eeie}, *Echo*	42^{eeie}	16	T	F
t + 44	52^{eeie}	46^{eeee}	36^{eeie}	46^{eeie}	16	T	F
t + 47	55^{eeie}	49^{eeee}	39^{eeie}	49^{eeee}	16	T	F
t + 48	~~56~~ → 50^{eeee}	50^{eeee}	40^{eeee}	50^{eeee}	10	T	F
t + 54	56^{eeee}	~~56~~ → 56^{eeee}	46^{eeee}	~~56~~ → 56^{eeee}	10	T	F
t + 64	66^{eeee}	66^{eeee}	~~56~~ → 66^{eeee}	66^{eeee}	10	T	F
t + 70	72^{eeee}, *Echo*	72^{eeee}, *Echo*	72^{eeee}, *Echo*	72^{eeee}, *Echo*	0	F	T
t + 78	80^{eeee}, *Echo*	80^{eeee}, *Echo*	80^{eeee}, *Echo*	80^{eeee}, *Echo*	0	F	T
t + 86	88^{eeee}, *Echo*	88^{eeee}, *Echo*	88^{eeee}, *Echo*	88^{eeee}, *Echo*	0	F	T
…	…	…	…	…	0	F	T

Matrices *M* and *T* at N_1 at *LocalTimer* = 7γ = 56 when all received *Init* messages are valid.

Matrix *M*

16	21	32	18
9	16	22	16
0	2	16	5
6	16	25	16

Matrix *T*

0	6	16	6
-6	0	10	0
-16	-10	0	-10
-6	0	10	0

Timeline of activities at N_1: 0 --- 6,6 -------- 16
Ignoring extreme values of 0 and 16, the adjustment Amount is: (6 + 6) / 2 = 6

$D_{12} = M(1,2) + M(2,1) / 2 = 15 * C$
$D_{13} = M(1,3) + M(3,1) / 2 = 16 * C$
$D_{14} = M(1,4) + M(4,1) / 2 = 12 * C$
$D_{23} = M(2,3) + M(3,2) / 2 = 12 * C$
$D_{24} = M(2,4) + M(4,2) / 2 = 16 * C$
$D_{34} = M(3,4) + M(4,3) / 2 = 15 * C$

Matrices *M* and *T* at N_1 at *LocalTimer* = 10γ = 80 when all received *Echo* messages are valid.

Matrix *M*

8	7	8	4
7	8	4	8
8	4	8	7
4	8	7	8

Matrix *T*

0	0	0	0
-0	0	0	0
-0	-0	0	0
-0	-0	-0	0

Timeline of activities at N_1: --- 0,0,0,0 ----
Ignoring extreme values of 0 and 0, the adjustment Amount is: (0 + 0) / 2 = 0

$D_{12} = M(1,2) + M(2,1) / 2 = 7 * C$
$D_{13} = M(1,3) + M(3,1) / 2 = 8 * C$
$D_{14} = M(1,4) + M(4,1) / 2 = 4 * C$
$D_{23} = M(2,3) + M(3,2) / 2 = 4 * C$
$D_{24} = M(2,4) + M(4,2) / 2 = 8 * C$
$D_{34} = M(3,4) + M(4,3) / 2 = 7 * C$

Appendix B

Matrices *M* and *T* at N_1 at *LocalTimer* = 7γ when all received *Init* and *Echo* messages are valid.

Matrix *M*

16	21	32	18
9	16	22	16
0	2	16	5
6	16	25	16

Matrix *T*

0	6	16	6
-6	0	10	0
-16	-10	0	-10
-6	0	10	0

$D_{12} = M(1,2) + M(2,1) / 2 = 15 * C$
$D_{13} = M(1,3) + M(3,1) / 2 = 16 * C$
$D_{14} = M(1,4) + M(4,1) / 2 = 12 * C$
$D_{23} = M(2,3) + M(3,2) / 2 = 12 * C$
$D_{24} = M(2,4) + M(4,2) / 2 = 16 * C$
$D_{34} = M(3,4) + M(4,3) / 2 = 15 * C$

Recovering Invalid Init

Matrices *M* and *T* at N_1 at *LocalTimer* = 7γ with some invalid entries (*Init* messages) but all *Echo* messages are valid, i.e., no faults during *Echo* exchange.

Matrix *M*

16	-	32	18
9	16	-	16
0	2	16	-
6	16	25	16

Matrix *T*

0	-	16	6
-	0	-	0
-16	-	0	-
-6	0	-	0

T(1,2) = T(1,4) - T(2,4) = 6 - 0 = 6, *T(2,1) = -T(1,2)* = -6
T(2,3) = T(1,3) - T(1,2) = 16 - 6 = 10, *T(3,2) = -T(2,3)* = -10
T(3,4) = T(1,4) - T(1,3) = 6 - 16 = -10, *T(4,3) = -T(3,4)* = 10
And *M* is readily restored using Equation 1.
For *K* = 4, three, i.e., *K*-1, simultaneous *Init* link faults were tolerated (recovered).

Recovering Invalid Echo

Matrices *M* and *T* at N_1 at *LocalTimer* = 7γ with some invalid entries in *Init* and *Echo* messages, specifically, given 4 nodes and allowing for one fault per stage.

Matrix *M*

16	21	32	18
9	16	-	16
0	2	16	5
-	-	-	-

Matrix *T*

0	6	16	-
-6	0	-	-
-16	-	0	-
-	-	-	-

T(2,3) = T(1,3) - T(1,2) = 16 - 6 = 10, *T(3,2) = -T(2,3)* = -10

From Equation 1, *M(2,3)* = 22

Matrix *M*

16	21	32	18
9	16	22	16
0	2	16	5
-	-	-	-

Matrix *T*

0	6	16	-
-6	0	10	-
-16	-10	0	-
-	-	-	-

Note N_4 did not broadcast *Echo* message to N_1.

V = M(1,4) = (18, 16, 5)

Using *V*, D_{ij}, and trilateration, timing of N_4 in *T* is restored. *M* is subsequently restored using Equation 1.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
01-07-2017	Technical Memorandum	

4. TITLE AND SUBTITLE

An Autonomous Distributed Fault-Tolerant Local Positioning System

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

999182.02.85.07.01

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

NASA Langley Research Center
Hampton, VA 23681-2199

8. PERFORMING ORGANIZATION REPORT NUMBER

L-20782

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-2017-219638

12. DISTRIBUTION/AVAILABILITY STATEMENT

Unclassified
Subject Category 62
Availability: NASA STI Program (757) 864-9658

13. SUPPLEMENTARY NOTES

14. ABSTRACT

We describe a fault-tolerant, GPS-independent (Global Positioning System) distributed autonomous positioning system for static/mobile objects and present solutions for providing highly-accurate geo-location data for the static/mobile objects in dynamic environments. The reliability and accuracy of a positioning system fundamentally depends on two factors; its timeliness in broadcasting signals and the knowledge of its geometry, i.e., locations and distances of the beacons. Existing distributed positioning systems either synchronize to a common external source like GPS or establish their own time synchrony using a scheme similar to a master-slave by designating a particular beacon as the master and other beacons synchronize to it, resulting in a single point of failure. Another drawback of existing positioning systems is their lack of addressing various fault manifestations, in particular, communication link failures, which, as in wireless networks, are increasingly dominating the process failures and are typically transient and mobile, in the sense that they typically affect different messages to/from different processes over time.

15. SUBJECT TERMS

Autonomous; Distributed; Dynamic; Fault-tolerant; Geo-location; Independent; Mobile; Positioning system; Synchronization

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)
U	U	U	UU	33	19b. TELEPHONE NUMBER *(Include area code)* (757) 864-9658