A Fault-Tolerant Clock Synchronization and Geometry Determination Protocol

Mahyar R. Malekpour*
Langley Research Center, Hampton, VA

A fault-tolerant distributed protocol (algorithm) is presented that achieves optimum timing precision (clock synchronization) among the nodes and, simultaneously, determines the network's geometry (shape)—locations and distances of the nodes relative to each other—in a wireless distributed system. This protocol is based on the assumption of initial coarse synchrony of nodes' local clocks. The proposed solution assumes no prior knowledge of the nodes' locations, the distances between the nodes, or network's geometry, but assumes an ordered geometry where nodes have unique identifiers. This protocol accommodates large variations in the communication latencies among the nodes; thus, it applies equally to both wireless and wired networks.

Keywords: Fault tolerant, synchronization, distributed, positioning system, mobile, dynamic environment, multilateration

Nomenclature

D	=	minimum communication event-response delay
d	=	network communication imprecision
γ	=	$D + d$
F	=	maximum number of faults in the network
K	=	number of nodes in the network
i	=	$1..K$
N_i	=	ith Node
π	=	synchronization precision
C	=	convergence time
P_{LT}	=	synchronization period
Init	=	communication message
Echo	=	communication message

I. Introduction

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 (network level) coordinated operations with ultra-reliability. As a result, robust clock synchronization has become a required fundamental component of fault-tolerant safety-critical distributed systems. Clocks are typically modeled as local counters, which increase/decrease with a given rate according to real time. Synchronization of a distributed system is the process of **achieving** and **maintaining** a bounded skew among independent local clocks at the participating nodes. Synchronization has practical significance as a fundamental service for higher-level algorithms which solve other problems. For example, in safety-critical TDMA (Time Division Multiple Access) architectures[1,2,3,4], synchronization is the most crucial element of these systems. Another example is in Local Positioning Systems (LPS) where a network of three or more signaling beacons (nodes) is 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 is at each beacon and the higher the precision

* Aerospace Technologist, NASA LaRC, 1 S. Wright St. MS130, Hampton, VA, 23681-2199, senior AIAA.

across the network, the more accurate the estimated position at the receivers. Similarly, the more accurate the geometry is and knowledge of the beacons' location and distances from each other, the more accurate the estimated position at the receivers will be.

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 remains in the legitimate state. For clock synchronization, a legitimate state is a state where all parts (local clocks) in the system are in synchrony, i.e., within a bounded skew of each other[5]. Typically, the assumed network topology is a regular graph such as a fully connected graph or a ring, since they provide a base case to solve the distributed synchronization problem.

A fundamental property of a robust distributed system is the capability of tolerating and potentially recovering from failures (loss of service due to a fault), which are not predictable in advance. A *fault* is a defect or flaw in a system component resulting in an incorrect state[2,6]. The requirement to handle faults adds a new dimension to the complexity of the synchronization of fault-tolerant distributed systems. In the context of fault-tolerant distributed systems, a fault presenting different symptoms to different observers is known as Byzantine (arbitrary) fault. To prevent single point failure, LPSs that are used as an alternate, or to complement GPS (Global Positioning System), need to establish their time synchrony and geometry internally and without reliance on an external source or a designated source internal to the network (master-slave scheme).

We call an approach to solving the clock synchronization problem *direct* if it relies solely on local (node level) detection and filtering of faults. This approach is primarily limited to detecting timing and/or value faults of a node's incoming messages. In contrast, we call an approach *indirect* if it relies on the global (network level) detection and filtering of faults independent of, and in addition to, the local detection and filtering of the faults. This approach however requires coordination at the network level.

Thus far, there is no formally verified, direct, and deterministic solution for the general case of the clock synchronization problem. Furthermore, most attempts have been in trying to solve this problem directly, although, some approached to solve this problem indirectly using authenticated (signed) messages[7]. Driscoll *et al.* in Ref. 8, however, argues that: "While the arguments of unforgeable signed messages make sense in the context of communicating generals, the validity of necessary assumptions in a digital processing environment is not supportable. In fact, the philosophical approach of utilizing cryptography to address the problem within the real world of digital electronics makes little sense. The assumptions required to support the validity of unbreakable signatures are equally applicable to simpler approaches (such as appending a simple source ID or a CRC to the end of a message). It is not possible to prove such assumptions analytically for systems with failure probability requirements near 10^{-9}/hr." Furthermore, we believe, to be generally useful, algorithms that solve clock synchronization problem must be able to handle non-authenticated messages.

Also, addressing network element imperfections, such as oscillator drift with respect to real time and differences in the lengths of the physical communication media, is necessary to make a solution applicable to realizable systems. In Ref. 9, 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. Also, in wired systems, network's geometry is a non-issue since the nodes are physically connected to each other and the lengths of communication links do not change. In wireless systems, however, network's geometry plays a crucial role as the link failures are more frequent and typically transient and mobile. Unlike wired networks, a wireless network does not have to be static. Mobility, however, adds yet another dimension to the complexity of this problem even in the absence of faults. If the nodes are mobile, communication links vary with time and the network's shape changes dynamically as a function of time.

In Ref. 10, we defined direct and indirect approaches for solving the clock synchronization problem based on local and global detection and filtering of faults. We also presented a two-step strategy for solving the clock synchronization problem indirectly by first converting any message to a symmetric message and then using a verified symmetric-fault-tolerant protocol to synchronize the network. A symmetric-fault-tolerant protocol was also introduced in Ref. 10 (listed in Appendix A), that guarantees achieving synchrony. The network precision obtained, however, is a function of the communication delays, coarse-grained and less than ideal for practical purposes. To augment the symmetric-fault-tolerant protocol, in this paper, we introduce a protocol that guarantees fine-grained, optimum synchrony in only one iteration, assuming the network is initially coarsely synchronized. Synchronization precision, π, is the guaranteed upper bound on differences of the nodes' local clocks. In distributed systems, the theoretical limit of π is one clock tick. We refer to a network with precision of one clock tick as being fine-grained/finely synchronized. When the network precision is much more than this optimum value but far less than

the synchronization period, P_{LT} (formally defined in the next section), we refer to it as coarse-grained/coarsely synchronized.

There exist other clock synchronization algorithms (protocols) based on the assumption of initial synchrony, e.g., Ref. 11, 12, and 13. These algorithms achieve their optimum synchrony in multiple rounds of iterations and, although not explicitly stated, the underlying assumption is a static wired network. By accommodating larger variations in the communication latencies among the nodes (beacons), our solution equally applies to both wireless and wired networks. Although we assume initial coarse synchrony, unlike the algorithms in Ref. 11, 12, and 13, our solution achieves optimum synchrony in only one iteration for static and/or dynamic networks.

The requirement of initial knowledge of the network's geometry would impose restrictions on the applications and preclude scenarios where the nodes are mobile. To accommodate LPS applications, for instance, our proposed solution does not assume a prior knowledge of the nodes' locations, the distances between the nodes, or the network's geometry (shape). As a result, our solution lends itself to high-dynamic systems by accommodating necessary services for UAVs (unmanned aerial vehicles) to maneuver at high speed, and in dynamic and mobile environments. We, however, assume an ordered geometry where nodes have unique identifiers and the network topology is a fully-connected graph. Extending this solution to include other topologies is left for future work.

This paper is organized as follows. In Section II we provide a system overview. In Section III we present the protocol and provide examples in Appendix B to help with the understanding of our proposed solution. Finally, we conclude with remarks in Section IV.

II. System Overview

We consider synchronous message-passing 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 communication links (edges) representing the nodes' interconnectivity. The underlying topology considered is a fully connected network of K[†] nodes that exchange messages. For a fully connected graph (network) of K nodes, a wired network consists of $K(K-1)$ unidirectional (one-to-one) communication links while a wireless network consists of K unidirectional (one-to-many, i.e., broadcast) communication links. The systems considered consist of a set of good nodes/links and a set of faulty nodes/links. A good node is assumed to be an active participant and correctly execute the protocol (algorithm). A faulty node is either benign (detectably bad), symmetric, or bounded-arbitrary (Byzantine) faulty. We assume a maximum of F Byzantine faults present. We assume $F < K/3$ and define the minimum number of good nodes in the system as G, noting that $G > 2K/3$. For fully connected graphs the minimum number of nodes needed to maintain synchrony is well established to be $3F+1$ [7,14,15]. We leave the generalization to other topologies for future work.

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. We associate a link fault to its source node. 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 (one-to-many). 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[1]. But a broadcast message is assumed to arrive within a bounded delay at the destination nodes as described in the next section.

A. Communication Delay

The communication delay between directly connected (adjacent) nodes is expressed in terms of the minimum event-response delay, D, network imprecision, d, and communication delay, γ. These parameters are measured at the network level as shown in Figure 1.

As depicted in this figure, a message broadcast by a node N_1 at real time t_0 is expected to arrive at its directly connected adjacent nodes N_2 (first node) and N_3 (last

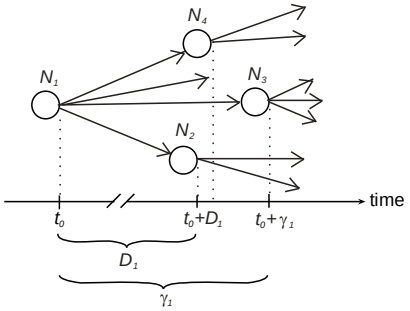

Figure 1. Event-response delay, D, and communication delay, γ.

[†] Since we use N_i to address a node, we use K here instead of n as is traditionally used in the literature.

node), processed, and subsequent messages generated by those nodes within the time interval of $\delta = [t_0+D, t_0+\gamma]$, where $D = min(D_i)$ and $\gamma = max(\gamma_i)$, for all $i = 1..K$. Communication between independently-clocked nodes is inherently imprecise. The network imprecision, $d = \gamma - D$, is due to many factors including, but not limited to, oscillators' drift with respect to real time—hence, variation in nodes' processing time—jitter, discretization error, temperature effects and differences in the lengths of the physical communication media. These parameters are assumed to be bounded, $D > 0$, $d \geq 0$, $\gamma > 0$, and have units of real-time clock ticks and their values are known in the network. In other words, we assume synchronous communication and bound the communication delay between any two directly connected adjacent nodes by $[D, \gamma]$. Thus, a broadcast message from a source node is assumed to arrive at the destination nodes within d of each other.

B. Protocol Messages

For the protocol presented here, the nodes communicate by exchanging *Init* and *Echo* messages. To maintain consistency with terminology used in the literature, we use the terms *Init* and *Echo* for messages as in Ref. 13; however, in our protocol, the *Echo* message is a vector of locally time-stamped events. We assume physical-layer error detection is dealt with separately and a node uses its own message γ clock ticks after its broadcast.

C. 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 $F < K/3$ 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 d each other, where $\gamma = D + d$
6. The network is coarsely synchronized with an initial precision of π_{init} , where $\pi_{init} \leq 2\gamma << P_{LT}$
7. Physical-layer error detection is dealt with separately

D. Self-Stabilization Properties

The following symbols are used in stating the subsequent 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, $P_{LT} >> 0$.
- $\Delta_{Net}(t)$, 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 $t \geq C$, $0 \leq \pi << P_{LT}$.
- π_{init}, the initial coarse synchronization precision, $\pi_{init} \leq 2\gamma << P_{LT}$.
- C, the convergence time, is defined as the bound on the maximum time for the network to achieve the guaranteed precision π.

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. Liveness: For all $t \geq C$, good node N_i, $i = 1..K$, there exists $(P_{LT} - \pi - \gamma) \leq U \leq P_{LT}$, such that $N_i.LocalTimer(t+1) = \text{mod}(N_i.LocalTimer(t)+1, U)$

The symmetric-fault-tolerant protocol in Ref. 10 provides coarse-grained synchrony with a guaranteed initial precision of $\pi_{init} = d + \gamma + \delta(d+\gamma) < 2\gamma$ clock ticks. In the context of this paper and the symmetric-fault-tolerant protocol, 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 $0 < \gamma << P_{LT}$, and the *LocalTimer* is reset after reaching P_{LT} (worst-case wraparound), a trivial solution is not possible.

In Section III, we introduce a clock synchronization protocol that achieves optimum precision of one clock tick, which is the theoretical limit.

E. What Self-Stabilization Properties Mean

The *Convergence* and *Closure* properties address achieving and maintaining network synchrony, respectively. 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 *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.

III. The Protocol

The protocol below is executed by all nodes N_i, every clock tick. This protocol is based on the assumption of initial coarse synchrony ($\pi_{init} << P_{LT}$). To maintain consistency with terminology used in the literature, we use the terms *Init* and *Echo* for messages as in Ref. 13; however, in our protocol, the *Echo* message is a vector of locally time-stamped events.

The protocol presented in Figure 2 starts executing when triggered by another algorithm (e.g., the symmetric-fault-tolerant protocol), which indicates that the network is coarsely synchronized. The integration process of this protocol with the symmetric-fault-tolerant protocol[10] is described in detail in Ref. 16. The following parameters are used in describing the protocol:

- $\omega = \pi_{init} + \gamma$
- ψ = *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 their broadcasts.

```
if (LocalTimer = ψ)
        Broadcast Init
if (LocalTimer = ω + ψ)
        Broadcast Echo
if (LocalTimer = 2ω + ψ)
        Recover()
        Adjust()
```

Figure 2. The Fault-Tolerant Clock Synchronization and Geometry Determination Protocol.

We now describe the *Recover()* 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.

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

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 $T(j,i)$.

Let D_{ij} be determined by the following equation:

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

This value is unknown if the right-hand side contains an invalid value. Note that D_{ij} will be the actual distance between N_i and N_j upon the network achieving synchrony and adjusting the timing of nodes' exchanged messages as reflected in M. Nevertheless, we use current values of D_{ij} in the process of achieving the optimum precision (fine-synchrony) in the network and determining correct distances between the nodes.

A. *Recover()*

In this section we describe the *Recover()* function that, in turn, consists of two parts, recover invalid, or missing, *Init* and recover invalid, or missing, *Echo* messages. Matrix T is introduced to aid with the recovery of missing data (faults), provide fault-tolerance, and achieve the optimum precision given the initial contents of matrix M. Once faults are recovered, the optimum precision is achieved and the contents of matrix M is restored.

1. Recover Invalid Init

Recall that a fault is defined as no message or an invalid received message. A faulty/no *Init* message manifests itself 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 link 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. To help with better understanding of the algorithm and its functions, examples are provided in Appendix B. After the network has reached fine-grain synchrony and provided there is sufficient data in M, D_{if} is determined using trilateration[‡] and the available 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.

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

Since a missing entry in $M(i,j)$ is synonymous to a link fault, 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 an *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 for future work.

2. Recover Invalid Echo

A faulty *Echo* message manifests itself as an invalid row in matrix M. Let, at node 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. In other words, 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 iterations 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.

[‡] 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.

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

B. *Adjust()*

The purpose of this function is to adjust the nodes' local times 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). To tolerate *F* faults, given the assumptions hold, we discard *F* values from both extremes. Although the reference point in time can be anywhere on the timeline, we choose the midpoint of the two remaining extreme 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, *LocalTimer*.

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

C. Proof of the Protocol

In this section we present a sketch of a proof of the protocol described in Fig. 2. When the assumptions hold, proofs of the *Recover()* and *Adjust()* functions are trivial.

Lemma Correctness – *The protocol in Fig. 2 achieves optimum precision.*

Proof – Given the initial precision, π_{init}, is known and the nodes broadcast their *Init* messages within π_{init} of each other, at time $t = \omega + \psi$, at least γ clock ticks have passed since the last broadcast of an *Init* message and all nodes (modulo some experiencing faults manifested as bad/missing *Init* messages) will have received the *Init* messages. In a similar argument, at time $t = \omega + \psi$ all nodes broadcast their *Echo* messages within π_{init} of each other. At time $t = 2\omega + \psi$, at least γ clock ticks have passed since the last broadcast of an *Echo* message and all nodes (modulo some experiencing faults manifested as bad/missing *Echo* messages) will have received the *Echo* messages. If there exists faults (missing *Init* and/or *Echo* messages) and assuming the assumptions are not violated, at $t = 2\omega + \psi$, the nodes recover the missing *Init* and *Echo* messages via the *Recover()* function. The recovery process guarantees that the nodes have similar data when evoking the *Adjust()* function; thus ensures that the adjustment realigns their local clocks to the optimum precision. □

IV. Conclusion

We have presented a distributed fault-tolerant protocol that performs two distinct functions in a wireless network of $F < K/3$ nodes (F is the maximum of F Byzantine faults), given the network starts with a coarse-grain synchrony. First, it achieves fine-grained synchrony—optimum timing of one clock tick—in the presence of up to F Byzantine faulty nodes. Second, it determines the network's geometry without the initial knowledge of the nodes' locations or the distances between the nodes, provided the network's geometry is ordered, nodes have unique identifiers, and the network topology is a fully-connected graph. This protocol lends itself to mobile, high-dynamic environments where the network is not required to be bounded to remain physically stationary or to retain static geometry.

References

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

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

[3]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.

[4]Steiner, W.; Dutertre, B.: "Automated Formal Verification of the TTEthernet Synchronization Quality," *3rd NASA Formal Method Symposium*, April 2011.

[5]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.

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

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

[8]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.

[9]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.

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

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

[12]Welch, J.L., Lynch N.: "A New Fault-Tolerant Algorithm for Clock Synchronization," *Information and Computation 77(1)*, pp. 1-36, 1998.

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

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

[15]Dolev, D.; Halpern, J.Y.; Strong, R.: "On the Possibility and Impossibility of Achieving Clock Synchronization," *proceedings of the 16th Annual ACM STOC*, pp. 504-511, 1984.

[16]Malekpour, M.R.: "An Autonomous Distributed Fault-Tolerant Local Positioning System," NASA/TM-2017-219638, July 2017.

Appendix A

The symmetric-fault-tolerant protocol, presented in Figures 1 and 2, is executed by all good nodes and consists of a synchronizer and a set of monitors that execute once every local clock tick. Four concurrent *if* statements describe the synchronizer. The function *ValidateMessage()* describes the monitor.

- 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, $P_{ST} >> 0$.

***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. Protocol functions.

Synchronizer:
ST1: if (*StateTimer* < 0) or (*Accept()*)
        StateTimer := 0, *// reset*

ST2: elseif (*StateTimer* < P_{ST})
        StateTimer := *StateTimer* + 1.

LT1: if (*LocalTimer* < 0) or
        (*LocalTimer* ≥ P_{LT}) or
        (*StateTimer* = $\lceil \pi_{init} \rceil$)
        LocalTimer := 0, // reset
LT2: else
        LocalTimer := *LocalTimer* + 1.

TT1: if (*TransmitTimer* < 0) or
        ((*TransmitTimer* ≥ γ) and
        (*StateTimer* ≥ P_{ST}))
        TransmitTimer := 0,
TT1: elseif (*TransmitTimer* < γ)
        TransmitTimer := *TransmitTimer* + 1.

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

Monitor:
        ValidateMessage().

Figure 2. The symmetric-fault-tolerant protocol.

Appendix B

The following is an example of a fully connected graph consisting of 4 nodes, where $F = 0$. Additional examples are provided in Ref. 16 to give the reader a quick review of and help in understanding the behavior of the Symmetric-Fault-Tolerant protocol and the Fault-Tolerant Clock Synchronization and Geometry Determination protocol, as well as the Fault-Tolerant Integrated Self-Synchronizing algorithm that assimilates the two protocols together.

System parameters:
D = 4 clock ticks, d = 4 clock ticks → γ = 8 clock ticks
K = 4 nodes, G = 4 nodes, F = 0 nodes
ψ = *ResetLocalTimerAt* = γ = 8 clock ticks
$\pi_{init} = d + \gamma + \delta(d + \gamma) \leq 2\gamma$ → π_{init} = 16 clock ticks (worse case)
$\omega = \pi_{init} + \gamma = 3\gamma = 24$

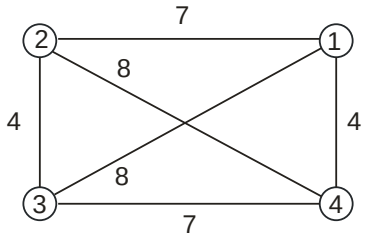

Figure 1. A 4-node network.

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

Table 1. Matrix *M*

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

Table 2. Matrix *T*

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

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

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

Table 3. Matrix *M*

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

Table 4. Matrix *T*

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

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

Recover 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.

Table 5. Matrix *M*

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

Table 6. 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 $K = 4$, $K-1 = 3$, simultaneous link faults were tolerated (recovered).

Recover 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.

Table 7. Matrix *M*

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

Table 8. 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, $M(2,3) = 22$

Table 9. Matrix *M*

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

Table 10. 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.