Global Clock Synchronization in Sensor Networks

Qun Li, *Member*, *IEEE*, and Daniela Rus, *Member*, *IEEE*

Abstract—Global synchronization is important for many sensor network applications that require precise mapping of collected sensor data with the time of the events, for example, in tracking and surveillance. It also plays an important role in energy conservation in MAC layer protocols. This paper describes four methods to achieve global synchronization in a sensor network: a node-based approach, a hierarchical cluster-based method, a diffusion-based method, and a fault-tolerant diffusion-based method. The diffusion-based protocol is fully localized. We present two implementations of the diffusion-based protocol for synchronous and asynchronous systems and prove its convergence. Finally, we show that, by imposing some constraints on the sensor network, global clock synchronization can be achieved in the presence of malicious nodes that exhibit Byzantine failures.

Index Terms—Sensor networks, fault tolerance.

1 Introduction

Many emerging sensor network applications require that the sensors in the network agree on the time. A sensor system with global clock will be capable of coordinated operation and data synthesis for future predictions. Consider, for example, a vehicle tracking application. Each sensor may know the time when a vehicle is approaching. By matching the sensor location and sensing time, the sensor system may predict the vehicle moving direction and speed. Without a global agreement on time, the data from different sensors cannot be matched up. Most applications that require the coordination of locally sensed data (e.g., environment monitoring) or coordination of mobile nodes (e.g., localization in the presence of mobility) are facilitated by the ability of the system to achieve global clock synchronization.

Clock synchronization has been a seminal topic in distributed systems [9], [13], [21], [18], but extending these results and designing clock synchronization algorithms in the context of a sensor network is challenging for several reasons. First, traditional distributed systems assume that all the nodes in a network can communicate directly with each other. A sensor network, however, is subject to spatial constraints. Nodes only communicate directly with their neighbors. Communication between two remote nodes is accomplished by message relay using intermediate nodes. Second, nodes in a sensor network generally rely on less information about the system than more traditional distributed systems, where nodes have access to the clock values of all the other members of the system, including the faulty nodes. Third, a sensor node has only limited processing capability. The computation-intensive signature algorithms, such as RSA, are not suitable for sensor networks. Instead, some light-weight algorithms (such as using a one-way key chain or a key management scheme) are more suitable. The spatial constraints, the communication cost and delay, and the diminished computational capability are key reasons why localized algorithms that involve lightweight computations are preferred for sensor networks.

This paper aims to explore clock synchronization in the context of the sensor network paradigm. We discuss four new methods for global synchronization in a sensor network called

1. the all-node-based method,
2. the cluster-based method,
3. the fully localized diffusion-based method, and
4. the fault-tolerant diffusion-based method.

The all-node-based method assumes the transmission time of a packet across a hop is the same for all nodes. A packet is transmitted around a cycle composed of all the nodes in the network. The packet transmission time is then amortized across the cycle. This method does not scale well because it requires the nodes in the entire network to participate in the synchronization process at the same time. To address scalability, we introduce a hierarchical method. Clusters are used to organize the whole network. The cluster heads are synchronized using the first method. A second synchronization round synchronizes the members within each cluster with their cluster head. Although the all-node-based method and the cluster-based method are not localized (because they involve all the nodes in the system), they have some interesting properties with respect to reducing the synchronization error.

The third protocol is a fully localized diffusion-based method with both synchronous and asynchronous implementations in which each node exchanges and updates information locally with its neighbors. No global operations are required. In the synchronous rate-based algorithm, neighboring nodes exchange clock reading values

- *Q. Li is with the Department of Computer Science, College of William and Mary, Williamsburg, VA 23187-8795. E-mail: liqun@cs.wm.edu.*
- *D. Rus is with the Computer Science and Artificial Intelligence Lab, Massachusetts Institute of Technology, Cambridge, MA 02139. E-mail: rus@csail.mit.edu.*

Manuscript received 21 July 2004; revised 1 May 2005; accepted 24 Aug. 2005; published online 21 Dec. 2005.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0244-0704.

0018-9340/06/$20.00 © 2006 IEEE Published by the IEEE Computer Society

proportional to their clock difference in a set order. A more practical variant is the asynchronous implementation of the diffusion-based approach in which a node can synchronize with its neighbors at any time in any order. The asynchronous algorithms can also adapt to limited node failure, adverse communication channel, and node mobility. The fault-tolerant diffusion-based protocol goes one step further in assuming the presence of malicious nodes that exhibit Byzantine faults.

The synchronization algorithms described in this paper can be extended to data aggregation problems in sensor networks, e.g., finding the average, highest, and lowest sensor data reading among all the sensors in the whole network. If the sensor data is attached to the message used by the all-node-based synchronization algorithm, the sum of the readings (thus the average reading), the highest, and lowest reading over the whole network can be computed post hoc or on the fly. Similarly, the diffusion-based algorithms can also be extended to perform these computations.

This paper is organized as follows: Section 3 presents the problem setup. Section 4 discusses the all-nodes synchronization scheme that requires all the nodes to participate in global synchronization upon a node's request. Section 5 describes the hierarchical cluster-based scheme that reduces the number of the participating nodes. Section 6 gives a synchronous diffusion-based algorithm that is fully localized. Section 7 discusses two asynchronous diffusion-based variants. Section 8 presents results from simulation experiments with the asynchronous averaging algorithm. Section 9 extends our diffusion algorithms to the case of Byzantine faults.

2 Related Work

Synchronization has been studied for a long time in traditional computer and embedded systems [22], [23], [20], [7], [26]. The classical paper on logical time [19] presented the solution to causal ordering of events in a distributed system. The classical work on clock synchronization in the presence of Byzantine fault in distributed systems includes [9], [13], [21], [18]. Lamport and Melliar-Smith [18] proposed using the nodes whose values are in the range of the middle one third to alleviate the influence of the malicious nodes. More recently, synchronization in sensor networks has also been studied [11], [10], [12], [27]. Elson and Romer [11] discuss the design principles for synchronization in sensor networks: use multiple, tunable modes of synchronization, avoid maintaining a global timescale for the entire network, use post-facto synchronization, adapt to application, and exploit domain knowledge. Elson et al. [10] propose a scheme called Reference-Broadcast Synchronization (RBS), in which a node sends reference broadcast beacons to its neighbors using physical-layer broadcasts. RBS gets around the nondeterminism of packet send time, access time, and propagation time, while depending only on the packet receive time. Since the packet receive time is the same for all receivers, this reference broadcast packet can be used to synchronize a set of receivers with one another. This scheme can also be extended to a multihop scenario. These two papers provide important practical building blocks for sensor network synchronization. However, they do not consider global synchronization over the entire network.

There has been some previous work on fault-tolerant sensor network design, including Krishnamachari and Iyengar's distributed Bayesian algorithms for fault-tolerant Event region detection in sensor networks [17]. Brooks et al. [6], [5], [4] were visionary in considering the sensor fusion problem 10 years ago. Many techniques used in these papers are useful and relevant to current sensor network research. For example, [6] surveys several algorithms on clock synchronization algorithms with Byzantine faults (in terms of approximate agreement and inexact agreement) and proposes a hybrid algorithm to fuse sensor data to one reading for all sensors in the presence of Byzantine faults. The proposed algorithm gives the best accuracy possible and increases the precision of the data in the distributed sensors based on the assumption that all sensors are connected and can communicate directly with each other. In this paper, we use a similar assumption: One third of the nodes should be good. However, we consider the problem in a modern sensor network scenario in which nodes can only communicate locally to their neighbors. The localized communication makes the problem much harder in that: 1) we have to compute a valid consensus locally and 2) the local consensus must be conveyed to other parts of the network; this is even harder because the relay nodes may be faulty or malicious.

The main idea of time-diffusion synchronization [28] is to start from a master node, adjust the clocks of its neighbors, and diffuse this clock adjustment to other nodes. This paper assumes no specific master nodes and diffusion nodes: Every node is a master node or a diffusion node in a broad sense. This property enhances the robustness of the algorithm. Furthermore, this paper also present an algorithm that explicitly addresses Byzantine failures. Karp et al. [16] proposed an optimal and global time synchronization using reference signals to synchronize the sensors. Unfortunately, the proposed method requires global information: The measured clock readings must be known to all nodes in order to compute the optimal values and the rates must be the same for all clocks. It is unclear if this method can be implemented in a localized way. Our work differs from time-diffusion synchronization and optimal and global time synchronization in that the proposed schemes here are fully localized and fault-tolerant.

Diffusion methods have been used in other applications, such as load balancing [8], [3], [30], [31], [1], [29]. Diffusion-based load balancing assumes the computation load on the computers is fine enough to be treated as a continuous quantity. By using diffusion, each computer can give its load to other connected computers or take load from connected computers if it is underloaded. Cybenko [8] and Boillat [3] analyzed the diffusion method in load balancing. They identified the sufficient and necessary conditions for convergence. The time complexity of the diffusion method was analyzed in [29]. Initial results about the asynchronous diffusion method are provided in [1], which gives the convergence proof of the asynchronous rate-based protocol, but not the average protocol. In this work, we prove the convergence of the asynchronous average protocol using a different method and show that the same method is easily tweaked for the convergence proof of the asynchronous rate-based protocol. These load balance algorithms fit into the robust interconnection network well. However, further investigation is needed to understand their application to sensor networks, where the communication channel is not perfect, nodes are prone to failure, and the system may be mobile.

Another consideration for the design of fault-tolerant protocols for sensor networks is sensor network security support. Work by Zhu et al. [32], [33] describes the basic security support for different sensor network applications. Work by [24] introduces the idea of using hierarchies for fault-tolerant protocols.

3 The Synchronization Problem

The time of a computer clock is measured as a function of the hardware oscillator $C(t) = k\int_{t_0}^{t} \omega(\tau)d\tau + C(t_0)$, where $\omega(\tau)$ is the angular frequency of the oscillator, k is a constant for that oscillator, and t is the time. The change of the value $C(t)$ dictates how events (or interrupts) can be captured.

The clocks in a sensor network can be inconsistent due to several reasons. 1) The clock may drift due to environment changes, such as temperature, pressure, battery voltage, etc. This has been a research topic in the operating system and Internet communities for many years. 2) The nodes in a sensor network may not be synchronized well initially, when the network is deployed. The sensors may be turned on at different times and their clocks may be running according to different initial values. 3) The clock can also be affected by the interaction of other components of the sensor system. For example, the Berkeley Mica Mote sensors may miss clock interrupts and the chance to increase the clock time value when they are busy handling message transmission or sensing tasks.

We explore how global synchronization can be achieved in sensor networks. We assume the hardware clock is not precise and nodes can read the current clock time and adjust the clock time at any time. The clock, however, has some granularity in its time reading (as coarse as a second or a fraction of a second), due to the hardware clock resolution or power conservation issues.[1] The sensor cannot determine the time elapsed in between the two ticks.

More specifically, we aim to provide coarse synchronization to many sensor network applications that need only low precision synchronization. Our goal is to synchronize the clocks in the whole network such that all the clocks have approximately the same reading at a global time point, irrespective of their relative distance. As a result, our proposed algorithms can be run in a less frequent fashion to alleviate the system load and, in turn, conserve energy.

4 All-Node-Based Synchronization

In this section, we describe a method to globally synchronize the clocks in a sensor network called *all-node-based synchronization*. This method is used on all the nodes in the system and it is most effective when the size of the sensor network is relatively small. In future sections of this paper, we describe ways to address scalability.

We assume the clock cycle on each node is the same. We believe this is a reasonable assumption since most sensors are programmed with the same parameters prior to deployment. We also assume the clock tick time is much longer than the packet transmission time.[2] Finally, we assume the message transmission time over each link and handling time on each node is roughly the same. This time can be obtained when the network traffic is small. That is, upon its initial deployment, a sensor network allows sufficient time solely for clock synchronization.[3]

The key idea is to send a message along a loop and record the initial time and the end time of the message. Then, by using the message traveling time, we can average the time to different segments of the loop and smooth over the error of the clocks. Algorithm 1 summarizes this method.

Algorithm 1 All-Node-Based Synchronization Algorithm in a Sensor Network

```
Find a ring that passes each node at least once that need to be synchronized (suppose the ring is composed of $k$ nodes)
A message is passed along the ring starting from an initiating node
Upon receipt of the message, each node records its current local time ($t_i$) and its order ($i$) in the ring. If the node receives messages more than once, it chooses one arbitrarily.
After the initiating node receives the message, it sends out another message informing each node on the ring the start time ($t_s$) and the end time ($t_e$) of the previous message
for each node, to adjust its local time $t$ do
    if $\exists m,\ m+1 \geq \frac{t_e-t_s+1}{k}\cdot(i-1) \geq t_i \geq \frac{t_e-t_s}{k}\cdot(i-1) \geq m$ then
        node $n_i$ adjusts its time to $t - t_i + t_s + m$
    if $\exists m,\ m+1 \geq \frac{t_e-t_s+1}{k}\cdot(i-1) \geq m \geq \frac{t_e-t_s}{k}\cdot(i-1) \geq m-1$ then
        node $n_i$ adjusts its time to $t - t_i + t_s + m$
```

Algorithm 1 times how long it takes to route a message along specified paths in the sensor system and uses this difference iteratively to correct the time for all the nodes along the path. In order to synchronize the entire network, paths need to be designed so that they contain all the nodes, but a specific node may appear multiple times. The synchronization is divided into two phases. In the first phase, a synchronization packet is sent along a ring. The initiating node of the packet records its local starting time and the ending time of the packet. Each other node simply forwards the packet and records how many hops the packet had traveled thus far. In the second phase, a clock correction packet is sent along the same ring. This packet informs each node of the packet starting and ending time for the initiating node and the total hops in the cycle. Each node then computes its clock adjustment.

Consider the example in Fig. 1. Node n_1 initiates clock synchronization and generates a message with its current time, t_s, attached. The time t_s is the exact tick time of

[1] A high frequency of clock ticks leads to a much higher power consumption; a reasonable frequency should be determined in a task-directed fashion.

[2] Higher clock frequency consumes more power, therefore, in many applications, the clock rate is set to be slow.

[3] During the synchronization time, all other messages except the synchronization messages are suppressed for transmission.

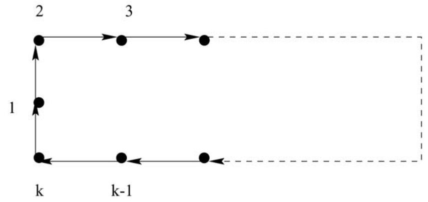


Fig. 1. A clock synchronization message traveling along a loop originated from node n_1 and then back to node n_1.

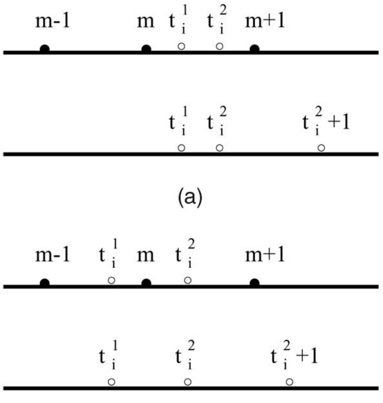


Fig. 2. Node n_i receives the synchronization message at time t_i such that $t_i^1 \leq t_i \leq t_i^2$. Since $|t_i^2 - t_i^1| < 1$, t_i^1 and t_i^2 are either in the range of two consecutive integers or separated by only one integer. Those two figures describe the two cases. On the second line in each figure, the next tick of n_i can be at any point between t_i^1 and $t_i^2 + 1$. At that point, it adjusts its time to $m + 1$.

node n_1's local clock. Node n_1 sends out the message to node n_2, node n_2 relays the message to node n_3, etc., finally, node n_k returns the message to the originator of the message, node n_1. The message may travel to a node more than once, that is, some nodes on this path may be visited repeatedly. Upon receipt of a message, each node keeps a record of the time of its clock and the time attached to the message when the message was created. Node n_1 gets the message start time t_s and the ending time t_e. It computes the difference $t_e - t_s$ and forwards it to the same node as for the previous message. The message will travel along the same path; this time, each sensor will try to get the number of hops to node n_1 (including node n_1) by putting a hops item in the message. For simplicity, we use the node id as the number of hops from node n_1. Each node then adjusts its clock as described in Algorithm 1. When a node appears twice in the cycle, it arbitrarily chooses its hops i from the two hop numbers and the corresponding t_i. In lines 7 and 9, a node adjusts its clock using different value of m obtained from lines 6 and 8, respectively.

Theorem 1. *After running Algorithm 1, the relative clock error between any two nodes is at most* 3Δ.

Proof. Without loss of generality, we consider the elapsed time as node n_1's time.

For any n_i, let the time when the first message arrives be t_i. The node will adjust its clock at the next tick after receiving the message. We know the time the message travels from n_1 to n_i is $t_i = \frac{t_e - t_s + \alpha}{k} \cdot (i-1)$, where $0 < \alpha < 1$ (because, when the message returns to n_1, the clock has already ticked time t_e), which is between $t_i^1 = \frac{t_e - t_s}{k} \cdot (i-1)$ and $t_i^2 = \frac{t_e - t_s + 1}{k} \cdot (i-1)$. The message arrival time is $t_s + t_i$. $t_i^2 - t_i^1 = \frac{i-1}{k} \leq 1$, so they must both be in the range of two integers $[m, m+1]$ or $t_i^1 \in [m-1, m]$, while $t_i^2 \in [m, m+1]$ (see Fig. 2).

1. If $m + 1 \geq \frac{t_e - t_s + 1}{k} \cdot (i-1) \geq t_i \geq \frac{t_e - t_s}{k} \cdot (i-1) \geq m$, n_i adjusts its next clock tick to $t_s + m + 1$. The next tick of n_i after $t_s + t_i$ and before $t_s + t_i + 1$ must be between $t_s + t_i^1$ and $t_s + t_i^2 + 1$ in n_1 time, which must be between $t_s + m$ and $t_s + m + 2$, so the maximal error to the time of n_1 is 1 tick time, that is, Δ.
2. If

$$m + 1 \geq \frac{t_e - t_s + 1}{k} \cdot (i-1) \geq m$$
$$\geq \frac{t_e - t_s}{k} \cdot (i-1) \geq m - 1,$$

n_i adjusts its next clock tick to $t_s + m + 1$. The next tick must be between $t_s + t_i^1$ and $t_s + t_i^2 + 1$, which must be between $t_s + m - 1$ and $t_s + m + 2$, so the maximal error to the time of n_1 is 2 tick time, that is, 2Δ.

For the first case, the error (the value obtained by subtracting n_1's time from n_i's time) is in the range of $[-\Delta, 0]$. For the second case, since $t_i^2 - t_i^1 = \frac{i-1}{k}$, we have $t_i^2 - m, \frac{i-1}{k}$ and $m - t_i^1, \frac{i-1}{k}$, thus $|t_i - m| \leq \frac{i-1}{k}$. The next tick must be between $t_s + m - \frac{i-1}{k}$ and $t_s + m + \frac{i-1}{k} + 1$, so the maximal error is

$$\left(t_s + m + 1 - \left(t_s + m - \frac{i-1}{k}\right)\right)\Delta = \left(1 + \frac{i-1}{k}\right)\Delta.$$

The error is in the range of $[-(1 + \frac{i-1}{k})\Delta, \frac{i-1}{k}\Delta]$.

Since the next tick of any sensor is between $m-1$ and $m+2$, the maximal error between any of two sensors is 3Δ.

To be more precise, we have the following: Since all the absolute errors (the value obtained by subtracting n_1's time from the n_i's time) are in the range of $[-\Delta, 0]$ or $[-(1 + \frac{i-1}{k})\Delta, \frac{i-1}{k}\Delta]$, the maximal error between any two nodes must be $(1 + \frac{2(i-1)}{k})\Delta$, which is obtained when one node has the error of $-(1 + \frac{i-1}{k})\Delta$ and another one has the error of $\frac{i-1}{k}\Delta$. □

If all the synchronizations are referenced to the same node, e.g., n_1, then the maximal error to that node is $[-(1 + \frac{i-1}{k})\Delta, \frac{i-1}{k}\Delta]$, where k is the minimal number of nodes synchronized in each synchronization round (excluding the reference node).

5 Cluster-Based Synchronization

The synchronization method proposed in Section 4 has a provable bound, but it requires all the nodes to participate in one single synchronization session. This can be mitigated using a hierarchical approach. More specifically, if the network can be organized into clusters, we propose to synchronize the whole network using Algorithm 2. In Algorithm 2, we first use the same method as in Algorithm 1 to synchronize all the cluster heads by designing a message path that contains all the cluster heads (we call them the initiators base). Then, in the second step, the nodes in each cluster can be synchronized with their head.

Algorithm 2 The Cluster Synchronization Algorithm

```
Run any clustering algorithm to organize the network into clusters
Synchronize the cluster heads with a base using Alg. 1
for each cluster do
    Synchronize the cluster members with the cluster head
```

This method can adapt to different clustering schemes. A cluster can be composed of the nodes within the transmission range of the cluster head; it can also be comprised of the nodes within some geographical area called a zone. For the first type of clustering, synchronization can be done with RBS. First, a reference broadcast is sent by the head to synchronize all the other cluster members. Then, any other node in the cluster sends out another reference broadcast to synchronize. The clock difference can be calculated with these two broadcasts and all the nonhead members can adjust their clocks according to the head's clock. In a zone clustering, we can use the same method as Algorithm 1 to first design a cycle that includes all the nodes of the cluster and synchronize them all. The head of the cluster will be the initiator of the intracluster synchronization.

Using Algorithm 1 in the two hierarchy stages increases the flexibility of the algorithm, but decreases the precision of the synchronization. Consider the base, n_0, a cluster head, n_i, and n_{ij}, a member of n_i's cluster. Suppose i is the order of n_i on the intercluster synchronization path of length k and j is the order of node n_{ij} on the intracluster synchronization path of length m. By the previous section, we have $t_i - t_0 \in [-(1+\frac{i-1}{k})\Delta, \frac{i-1}{k}\Delta]$ and $t_{ij} - t_i \in [-(1+\frac{j-1}{m})\Delta, \frac{j-1}{m}\Delta]$. Thus, the error $t_{ij} - t_0 \in [-(2+\frac{i-1}{k}+\frac{j-1}{m})\Delta, (\frac{i-1}{k}+\frac{j-1}{m})\Delta]$. The maximal error is 6Δ.

6 Synchronous Diffusion

6.1 Why Use Diffusion?

Our previous methods use global time information sent to all the nodes and are not scalable for very large networks. The initiating node may encounter failure and, thus, the approach is not fault-tolerant. The nodes that participate in the synchronization must execute the related code approximately at the same time, which may be too hard in a large system. We now introduce a diffusion method that is fully distributed and localized. Synchronization is done locally, without a global synchronization initiator. It can also be done at arbitrary points in time as opposed to the strict timing requirements of the previous methods.

The diffusion method achieves global synchronization by spreading the local synchronization information to the entire system. The algorithm can choose various global values to synchronize the network provided that each node in the network agrees to change its clock reading to the consensus value. An easy option is to choose the highest or lowest reading over the network. Synchronization to the highest or lowest value entails a simple algorithm. However, if we allow for faulty or malicious nodes, such a node may impose an abnormally high or low clock reading, which is likely to ruin the synchronization. To make the algorithms more robust and reasonable, the following algorithms use the global average value as the ultimate synchronization clock reading. The main idea of the algorithms is to average all the clock time readings and set each clock to the average time. A node with high clock time reading diffuses that time value to its neighbors and levels down its clock time. A node with low time reading absorbs some of the values from its neighbors and increases its value. After a certain number of rounds of diffusion, the clock in each sensor will have the same value.

We assume the existence of two basic operations: 1) the neighboring nodes compare their clock readings at a certain time point and 2) the nodes change their clock accordingly. This, however, may be a problem because the clock comparison and the clock update cannot be done simultaneously (especially when clock comparison may take several steps). The clock updates based on the clock readings of the comparison time will be incorrect. The solution is to ask each node to keep a record of how much time elapses after the clock comparison on each node and use this time in the clock update. For ease of explanation, our algorithms use the two operations with this implementation fix.

The diffusion synchronization method can be viewed as a high level framework for global synchronization. The low level implementations can be different as long as they provide a way to compare the clock difference among all the neighbors. For example, we can use the RBS scheme [10] as the low level component. When a node (say A) intends to do local synchronization, it sends out a reference broadcast to all its neighbors. The neighbors record the time of the reception time, so the clock differences among these neighbors can be computed. Next, a node that is the mutual neighbor of both A and any of A's neighbors (say B) sends out another reference broadcast so that the clock difference between A and B can be computed. By way of B, we can achieve the clock comparison among A and any of its neighbors. We can choose a different low-level component. For example, assuming fixed and known transmission time (say t_0) of a packet between neighbors, the clock reading exchange can be done by broadcasting a packet with the sender's current time. Upon receipt of the packet, each node records its local time (say t_2) and the packet sending time attached to the packet (say t_1). Thus, the clock difference between the sender and the receiver can be computed as $t_2 - t_1 - t_0$.

6.2 The Rate-Based Synchronous Diffusion Algorithm

We assume that we have n sensors in the system. This network is represented as a graph $G(V, E)$ in which the vertices are the sensors and the edge relationship is defined by the sensor communication connectivity. Each node has a corresponding vertex in the graph. If two sensors are within transmission range of each other, their corresponding vertices n_i and n_j have an edge to connect them.

Let $C = (c_1^t, c_2^t, \cdots, c_n^t)^T$ (where T denotes matrix or vector transposition) contain the time readings of the sensors in the network at time t, where c_i^t is the clock reading for sensor n_i at time t (for simplicity, use c_i). If n_i and n_j are within their transmission range and $c_i > c_j$, we want to decrease c_i and increase c_j. Since we are computing the average value for all the sensors in the system, the decrease of c_i should go to the increase of c_j to maintain the conservation law. Suppose that the diffusion value is proportional to $c_i - c_j$ and the diffusion rate is $r_{ij} > 0$. ($r_{ij} = 0$ if n_i and n_j are not neighbors. r_{ij} can be chosen randomly provided $\sum_{j \neq i} r_{i,j} \leq 1$.) Sensor n_i will lose some of its clock value, $r_{ij} \cdot (c_i - c_j)$, to sensor n_j and it will lose a total of $\sum_{j \neq i} r_{i,j} \cdot (c_i - c_j)$ to all its neighbors (or gain some value if that sum is negative). Its value will become $c_i - \sum_{j \neq i} r_{i,j} \cdot (c_i - c_j) = (1 - \sum_{j \neq i} r_{i,j}) \cdot c_i + \sum_{j \neq i} r_{i,j} \cdot c_j$

Algorithm 3 shows the diffusion method. Synchronization between a sensor and its neighbors is done by clock comparison and update operations. Because we only consider the time difference between two sensors instead of the absolute clock time value, it is not required that all the sensors must do this local synchronization at the same time. In line 6, the exchanged value between sensor n_i and its neighbor n_j is proportional to the time difference between them.

Algorithm 3 Diffusion algorithm to synchronize the whole network

```
1: for each sensor $n_i$ in the network do
2:     Exchange clock times with $n_i$'s neighbors
3:     for each neighbor $n_j$ do
4:         Let the times of $n_i$ and $n_j$ be $c_i$ and $c_j$
5:         Change $n_j$'s time to $c_j + r_{ij}(c_i - c_j)$
6:     Change $n_i$'s time to $c_i - \sum_{all\ n_i's\ neighbors\ n_j} r_{i,j} \cdot (c_i - c_j)$
```

The clock value diffusion formula can be described by applying the following matrix R on the clock reading vector:

$$R = \begin{pmatrix} r_{11}, & r_{12}, & \cdots, & r_{1n} \\ r_{21}, & r_{22}, & \cdots, & r_{2n} \\ \cdots, & \cdots, & \cdots, & \cdots \\ r_{n1}, & r_{n2}, & \cdots, & r_{nn} \end{pmatrix}.$$

In the matrix, $r_{ij} = r_{ji}$; $r_{ij} = 0$ if n_i and n_j are not neighbors, so

$$r_{ii} = 1 - \sum_{j,(i,j) \in E} r_{ij} = 1 - \sum_{j \neq i} r_{ij}.$$

Every time we run Algorithm 3, we apply matrix R to the clock reading vector. More precisely, $C^{t+1} = R \cdot C^t$. Let $C^o = (c_1^0, c_2^0, \cdots, c_n^0)^T$ be the initial clock reading distribution at time 0. We have $C^{t+1} = R^{t+1} \cdot C^0$. We hope this time reading vector will become $C^s = (c^0, c^0, \cdots, c^0)^T$ (where $c^0 = \sum_{k=1}^n c_k^0 / n$) after running the algorithm. We call C^s the synchronized clock distribution. It is easy to see that, when the sensors achieve C^s, this value is stable. Note that C^s is an eigenvector of matrix R with respect to eigenvalue 1.

6.3 Convergence of the Rate-Based Synchronous Algorithm

In this section, we show that Algorithm 3 achieves global synchronization in the entire network. More specifically, the time vector $C^{t+1} = R^{t+1} \cdot C^0$ converges to the synchronized clock distribution C^s. We first summarize the convergence result of our algorithm, which has the flavor of convergence of a Markov chain. We then discuss the convergence speed.

We assume the graph we derived from the network is strongly connected, so the matrix R is irreducible. R is also symmetric and positive because $r_{ij} = r_{ji} > 0$. The eigenvalues of a symmetric matrix are real. Using the Perron-Frobenius theorem [14], the matrix R has the following eigenvalues: $1 = \lambda_1 > \lambda_2 \geq \lambda_3 \geq \cdots \geq \lambda_n \geq -1$.

Theorem 2. *The convergence of Algorithm 3 depends on the eigenvalue of the second largest absolute value, that is, $\lambda_{max} = max(|\lambda_2|, |\lambda_n|)$. If $\lambda_{max} < 1$, the iteration will converge to the synchronized clock vector.*

Proof. Suppose $\lambda_{max} < 1$. Let R have n normalized independent eigenvectors $e_1, e_2, \cdots, e_n$ corresponding to the eigenvalues $\lambda_1, \lambda_2, \lambda_3, \cdots, \lambda_n$. Let $A = (e_1, e_2, \cdots, e_n)$. We have $R = A^{-1}DA$, where D is the diagonal matrix of eigenvalues $D_{ii} = \lambda_i$. $R^t = A^{-1}D^tA = \sum_{i=1}^n \lambda_i^t M_i$. R^t will approach M_1 if $\lambda_{max} < 1$. Since C^0 can be written as $\sum_{i=1}^n a_i e_i$, we have $RC^0 = \sum_{i=1}^n a_i \lambda_i e_i$ and

$$R^t C^0 = \sum_{i=1}^n a_i \lambda_i^t e_i.$$

The last term approaches $a_1 \lambda_1^t e_1$ (all the elements of e_1 are the same), that is, all the clock readings are the same eventually. □

Theorem 3. *If $\lambda_n = -1$, then there is a permutation matrix P such that $P^T RP$ has block partitioned form:*

$$\begin{pmatrix} 0 & R_{12} \\ R_{21} & 0 \end{pmatrix}.$$

Proof. If $\lambda_n = -1$, then there are exactly two eigenvalues $\lambda_1 = 1$ and $\lambda_n = -1$ of modulus 1[4] and there is a permutation matrix P such that $P^T RP$ has block partitioned form:

$$\begin{pmatrix} 0 & R_{12} \\ R_{21} & 0 \end{pmatrix}.$$

□

By examining the above matrix, we have $r_{ii} = 0$ for all $1 \leq i \leq n$ if $\lambda_n = -1$. Likewise, if $r_{ii} \neq 0$ for some i, we have $\lambda_n \neq -1$. In our algorithm, if we let $r_{ii} \neq 0$, the algorithm converges to the synchronized clock vector.

Now, consider the convergence speed of the matrix. We scale the sum of all the clock readings to 1, so $\sum_{i=1}^n c_i = 1$. We have $c^0 = 1/n$.

Theorem 4. *For R with eigenvalue $\lambda_{max} < 1$ and synchronized clock vector C^s, the relative error after running Algorithm 3 for t steps is $max_{i,j} \frac{r_{ij}^{(t)} - c^0}{c^0}$, where $r_{ij}^{(t)}$ is the $R^t(i, j)$.*

Proof. First, we have $max_{i,j} \frac{|r_{ij}^{(t)} - 1/n|}{1/n} \leq \frac{\lambda_{max}^t}{1/n} = n\lambda_{max}^t$ since the elements of A and A^{-1} are no greater than 1. Then,

[4] The eigenvalues of a symmetric matrix are always real.

$$\begin{aligned} c_i^t &= \sum_{j=1}^{n} r_{ij}^{(t)} \cdot c_j^0 = \frac{1}{n}\sum c_j^0 + \sum\left(r_{ij}^{(t)} - \frac{1}{n}\right)c_j^0 \\ &= c^0 + \sum\left(r_{ij}^{(t)} - \frac{1}{n}\right)c_j^0. \end{aligned}$$

Thus, the relative error $\frac{|c_i^t-c^0|}{c^0} = \frac{\sum(|r_{ij}^{(t)}-\frac{1}{n}|)c_j^0}{c^0} \le n\lambda_{max}^t$ (since $c_j^0 \le 1$). □

The value of λ_{max} determines the convergence speed of our algorithm. Next, we evaluate the value of λ_{max}.

Let S be a set of sensors, $C_S = \sum_{i\in S} c_i^s = \frac{|S|}{n}$ be the capacity of S, and $F_S = \sum_{i\in S, j\notin S} r_{ij}/n$. Define $\Phi_S = F_S/C_S$. The conductance of the diffusion procedure is defined as $\Phi = min_{S:C_S\le\frac{1}{2}}\Phi_S$. By Jerrum and Sinclair [15], $\lambda_2 < 1 - \Phi^2/2$. Thus, we can bound the convergence speed if we know Φ.

7 The Asynchronous Diffusion Methods

7.1 Asynchronous Diffusion Algorithms

In the previous section, we analyzed a synchronous version of our diffusion-based algorithm, proved its convergence, and bound its convergence speed. The synchronous algorithm is localized, but it requires a set order for all the node operations. In this section, we extend the diffusion synchronization algorithm to remove this constraint. We ensure that all the nodes can perform operations in any order as long as each node is involved in the operations with nonzero probability. Our method can be shown to converge.

We start with an asynchronous averaging algorithm (Algorithm 4), which gives a very simple average operation of a node over its neighbors. Each node tries to compute the local average value directly by asking all its neighbors about their values; it then sends out the computed average value to all its neighbors so they can update their values.

Algorithm 4 Asynchronous Averaging Algorithm in a Sensor Network

```
for each node $n_i$ with uniform probability do
    Ask its neighbors the clock readings (read values from
    $n_i$ and its neighbors)
    Average the readings (compute)
    Send back to the neighbors the new value (write values
    to $n_i$ and its neighbors)
```

We assume the average operation is atomic, that is, if a node is involved in two or more average operations, these operations must be sequenced. We also assume the network is connected.

Theorem 5. *The asynchronous average algorithm converges to C (where C is the average value).*

Proof. Let H_t and L_t be the highest and lowest value, respectively, of all sensors at time t. We note that: 1) H_t is nonincreasing over time t (by the average algorithm, since there is no value greater than H_t at time t, H_t cannot increase from that time on); 2) L_t is nondecreasing over time t by symmetry.

We know $H_t \ge C$ and H_t is nonincreasing according to 1). Letting the infimum of the series H_t be M, we have $lim_{t\to\infty} H_t = M \ge C$. Suppose $M \ne C$. We will derive a contradiction.

Consider the function $\epsilon(k) = \sum_{i=1}^{k} n^i\epsilon$. Choose ϵ such that $M - \frac{n^{n+1}-1}{n-1}\epsilon = M - \epsilon(n) = C$, where n is the number of sensors. For any k ($k = n, n-1, \cdots, 1$), define S_k^1 to be the set of sensors whose values are greater than $M - \epsilon(k)$ and S_k^2 to be the set of the rest of the nodes.

For ϵ, there must exist a time t such that $H_t < M + \epsilon$; also, at that time, there must be some node whose value is less than $C = M - \epsilon(n)$ because C is the average value.

We first consider a snapshot of the kth step ($k = n, n-1, \cdots, 1$) in our constructive proof. If an average operation only involves the nodes in S_k^1 (or S_k^2), those nodes are still in S_k^1 (or S_k^2) after the operation. However, if an average operation involves nodes in both S_k^1 and S_k^2 (and we know there must be an operation that involves nodes from both sets since the network is connected), these nodes have value less than $M - \epsilon(k-1)$ after the operation due to the following reason: At least one node is from S_k^2. Even if all the other nodes have the highest possible value $M + \epsilon$, the average value is at most $\frac{(M+\epsilon)(m-1)+M-\epsilon(k)}{m} < M - \epsilon(k-1)$ where m is the number of concerned nodes in this average operation. Then, at least $|S_k^2| + 1$ nodes will be in S_{k-1}^2 after that operation.

Starting from sets S_n^1 and S_n^2 at time t, we have $|S_n^2| \ge 1$ (recall that there must be some node whose value is less than $C = M - \epsilon(n)$). After the first average operation for nodes that are in S_n^1 and S_n^2, we have $|S_{n-1}^2| \ge 2$. After the first average operation on nodes in S_{n-1}^1 and S_{n-1}^2, we have $|S_{n-2}^2| \ge 3$. So, eventually, we have $S_1^2 = n$.[5] This contradicts that the infimum of H_t is M (i.e., $H_t \ge M$).

Therefore, we have $lim_{t\to\infty} H_t = C$. In the same way, we can prove that $lim_{t\to\infty} L_t = C$. Combining these two results, we have that all the values on the sensors converge to C. □

Theorem 6. *The asynchronous rate-based algorithm converges to the global average value.*

Proof. We prove this result using an approach similar to Theorem 5. Let the $min\, r_{ij} = r$ ($i \ne j$ and $r_{ij} > 0$) and change the occurrence of $\epsilon(k)$ in the above proof to $\epsilon(k) = (\frac{1}{r^{n+1-k}} - 1)\epsilon$. Notice, for any step k ($k = n, n-1, \cdots, 1$), we have the following:

$$\begin{aligned} &M + \epsilon - r_{ij}(M + \epsilon - (M - \epsilon(k))) \\ &\le M + \epsilon - r(M + \epsilon - (M - \epsilon(k))) \\ &= M + \epsilon - r(\epsilon + \epsilon(k)) \\ &= M + \epsilon - r(\epsilon + \frac{\epsilon}{r^{n+1-k}} - \epsilon) \\ &= M + \epsilon - \frac{\epsilon}{r^{n-k}} = M - \epsilon(k-1). \end{aligned}$$

Thus, it follows that the asynchronous rate-based algorithm converges. □

We now define the deduced graph of the sensor network after time t, $G_t(V, E)$, where V is the set of vertices representing the sensor nodes and $(n_i, n_j) \in E$ (for $n_i \in V$

[5] Note that S_1^2 is the set of nodes whose values are less than $M - \epsilon(1) = M - n\epsilon$.

and $n_j \in V$) if and only if n_i and n_j are involved in the same average operation[6] after time t. The sensor network is operation connected $G_t(V, E)$ is connected for any $t > 0$. Algorithm 4 assumes all the nodes perform the algorithm with uniform distribution. However, as long as the network is operation connected, the convergence result still holds by examining our proof. This includes the following three scenarios, provided the network is operation connected: 1) The network is connected and each node has a nonzero probability of being involved in an average operation after any time t, 2) not all the neighbors of a node respond to the average operation as in the cases of failed nodes or bad communication channel, and 3) the sensor nodes are mobile.

We now discuss the convergence speed of the asynchronous algorithms. Let the error to be defined as $\frac{H_t - C}{C}$, where H_t is the highest value at time t. We consider the case in which only one node has stimulus value H_0 initially, while others have value 0. In a real scenario, all the nodes may have nonzero values. We can view this case as n copies of networks that are running and each copy starts with some value on only one node. In each step, the same operations are applied to all the copies and the real network clock reading vector is the sum of the clock reading vectors of all n copies. Note that the convergence on the real network is no slower than that on the slowest among all the n copies. Therefore, to evaluate the worst convergence, we need to consider a network with only one node that has nonzero initial value; our simulation assumes this as well.

Suppose, at time t, the highest value is H_t and most of the other nodes have value very close to C. The part that is close to C will be kept on each node in the future; the $H_t - C$ part on the node with the highest value will be leveled down to the other nodes. It follows that, every time we increase the number of rounds of the algorithm running linearly, we get the $H_t - C$ part, i.e., the error, reduced exponentially. This observation is verified in our simulation.

Other variations of our algorithm can choose the highest value (or lowest value) among all the neighbors to be the synchronized clock reading. When each node executes the diffusion operation exactly once in each round, it takes $O(n)$ rounds for the highest (or lowest) value to propagate to the whole network. Thus, the convergence time is $O(n)$, where n is the number of nodes.

7.2 Discussion

Our method is independent of and can be built upon any local synchronization method. The error in our diffusion method depends on the error inherent to the local synchronization method. The convergence speed of our diffusion method is slow compared to that of a synchronization algorithm with an initiator. However, the diffusion is useful when only a coarse synchronization is required. The following discusses when the protocols are useful and how we can improve on them:

1. The local synchronization is more important than the precise global synchronization since, in most applications, events are processed locally. For example, in the vehicle tracking application, the sensors that detect the vehicle communicate with each other to collaboratively obtain the information about the vehicle such as moving speed, direction, and vehicle type. Although the global synchronization convergence takes longer, local diffusion is very fast because a small number of sensors are involved.
2. The clock reading distribution is uniform in various areas of the sensor field. Thus, the average clock readings of different areas are approximately the same. After the local diffusion, the synchronized clock value in an area is approximately the same as that of a remote area.
3. Our proof shows that the asynchronous algorithms converge with random node operations and with any probability distribution of executions upon all the nodes as long as each node always has the chance to be involved in the execution. Each node can run the asynchronous operations on the fly without knowing what other nodes are doing. This model can adapt to the changing network topology, node failure, adverse communication conditions, node mobility, etc.
4. One of the factors that affects the convergence speed is that all nodes have the same probability to execute the average operation. The operation of a node whose clock is similar to its neighbors does not contribute much to convergence. An improvement to reducing communication and speeding up convergence is to adaptively bias toward the nodes with large difference to their neighbors. In this way, the large difference may be amortized quickly. The probability can be chosen proportional to $\frac{D}{C'}$, where D is the maximal difference among a node and its neighbors and C' is the average value estimated at the node.

8 Simulation

We implemented the averaging synchronization algorithm (Algorithm 4) in simulation. We ran a series of scenarios with different network parameters. For each time slot (say one second), each node executes the average operation once although the order of the operations of all the nodes is randomized. In a real network, the frequency is specified for each node and each node needs to perform the operation once for each set time interval. We define a round to be the time for each node to finish the average operation in Algorithm 4 exactly once, so the number of rounds for the network to achieve some error threshold signifies the convergence speed.

For each experimental set of parameters, the simulation was executed several times using a randomly generated network topology. In each experiment, a stimulus was generated at a randomly chosen node and propagated to the whole network until the relative error was achieved. In Fig. 4, Fig. 5, and Fig. 6, the data points are drawn from the experiments and the curves are plotted with the average values for each experimental set. The simulation results are presented as follows:

[6] Or exchange operation in rate-based algorithm. For convenience in the following discussion, we only talk about the averaging algorithm. However, the later discussion applies to the rate-based algorithm as well.

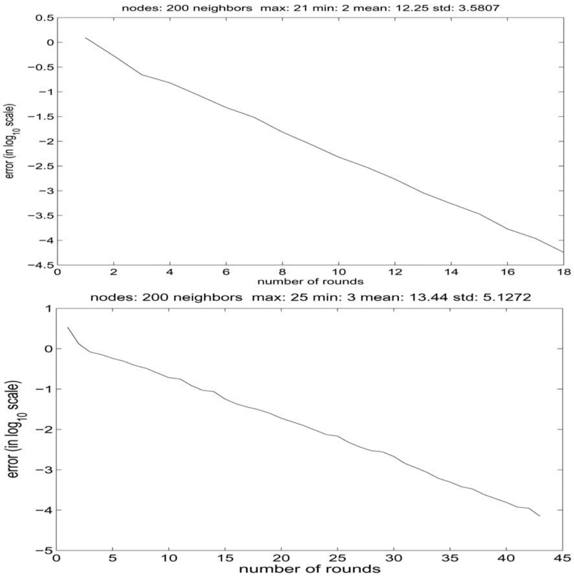


Fig. 3. Two typical simulations for the average algorithm. The experiments have the following parameters: number of nodes 200, sensor field 10x10, transmission range 1.5, and algorithm stop error 0.01 percent. The number of neighbors of each node varies in each experiment. In the first experiment, the maximal number of neighbors is 21, minimal number of neighbors is 2, average number is 12.25, the standard deviation is 3.58. In the second experiment, the numbers are 25, 3, 13.44, and 5.13, respectively.

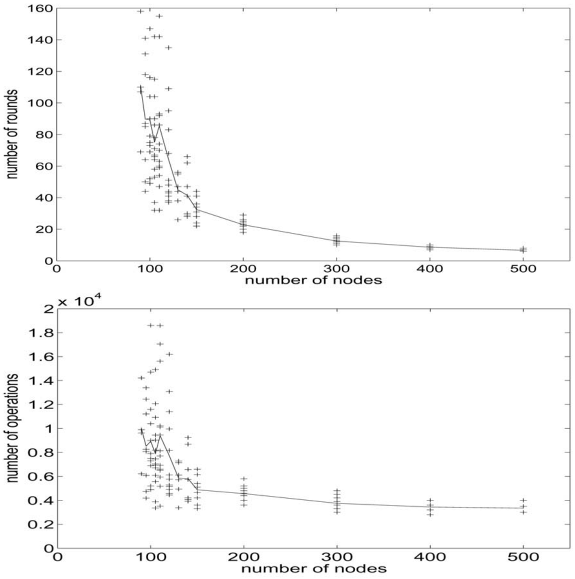


Fig. 4. The convergence speed with different number of nodes. The network parameters are: sensor field 10x10, transmission range 1.5, and algorithm stop error 0.01 percent. The markers are the number of rounds for each experiment. The plotted curve is the average number of rounds for one suite of network parameters. A sparse network with fewer nodes undergoes large variation in terms of convergence speed. The first figure shows the number of rounds for each network. The second figure presents the total number of operations conducted by all the nodes in each network.

1. The first suite of experiments (Fig. 3) relate the relative error with the number of rounds executed by the sensors. We generated two random networks with the the same set of network parameters. As we have conjectured in the previous section, the error rate decreases exponentially with the increase of the number of rounds. The simulation confirms our conjecture.
2. Convergence speed versus number of nodes with other parameters fixed (Fig. 4). The two figures evaluate the convergence speed with the number of nodes. Each data point in the figure represents a running on a randomly generated network. The markers are the number of rounds (in the first figure) and the number of total operations (in the second figure) for each experiment. The plotted curve is the average number of rounds and number of total operations for one suite of network parameters. A sparse network with fewer nodes undergoes large variation in terms of convergence speed.

   The first figure shows the number of rounds decreases with the increase of the number of nodes. The reason is that, when the number of neighbors of each node increases, the network is more connected, which expedites the diffusion. The second figure shows the total number of average operations conducted by all the nodes is approximately the same for each network. This is because the number of neighbors increases linearly with the number of nodes with other network parameters fixed.
3. Convergence speed versus transmission range with other parameters fixed (Fig. 5). The figure evaluates the convergence speed with different transmission ranges. The plotted curve is the average number of rounds for one suite of network parameters. With the same network scope and number of nodes, the convergence speed increases with the increase of the transmission range. This is due to the fact that the number of neighbors of each node increases and the number of nodes covered by each average operation increases.
4. Convergence speed versus number of nodes with fixed sensor density (Fig. 6). The plotted curve is the average number of rounds for one suite of network parameters. The convergence speed decreases linearly with the increase of the number of nodes in the context of the fixed sensor density. Since each node has the same number of neighbors

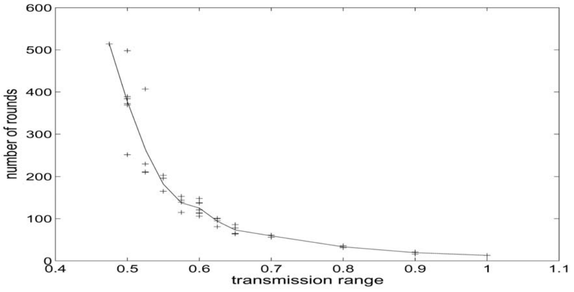


Fig. 5. The convergence speed with different transmission ranges. The network parameters are: sensor field 10x10, number of nodes 1,000, algorithm stops at error 0.01 percent. The crosses signify the number of rounds for each experiment. The plotted curve is the average number of rounds for networks with the same network parameters.

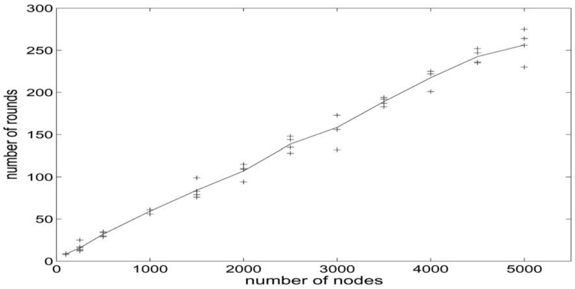


Fig. 6. The convergence speed with the number of nodes with fixed node density. The network parameters are: sensor field proportional to the number of nodes (10 nodes per unit area), transmission range 0.7, algorithm stops at error 0.1 percent. We see almost linear decrease in the convergence speed with the increase of the nodes. The plotted curve is the average number of rounds for networks with the same network parameters.

in each simulation, the diffusion speed is dependent on the number of sensors in the whole network.

5. Convergence speed compared with asynchronous rate-based protocol (Fig. 7). We plot the convergence speed for asynchronous rate-based protocol and average protocol. We find the convergence speed increases with a higher rate. The average protocol achieves approximately the same convergence speed as the rate-based for rate equals to 0.7.

9 Synchronization in the Presence of Malicious Nodes

The previous protocols assume all nodes are reliable and cooperative. We now consider an extension that allows some fraction of malicious nodes in the system. More specifically, we wish to design fault-tolerant algorithms for clock synchronization in the presence of Byzantine faults in a sensor network. That is, some nodes may behave arbitrarily in a malicious or unintentional way.

We consider a simple case in which some tamper-proof nodes (called N nodes) are introduced together with other normal nodes (called M nodes). A tamper-proof node will destroy itself once it is compromised. This guarantees that an N node will always be trusted. Tamper-proof nodes may be more expensive than the normal nodes, but future sensor networks are likely to have this kind of special and powerful nodes for hierarchical structure [24]. The N nodes can be less densely dispersed in a field than the normal nodes and they serve as cluster heads for the normal nodes. The role of an N node includes data aggregation, security key management, and other services. We make the following three assumptions: 1) Each normal node must belong to a cluster whose head is an N node, which means that each M node is one hop away from an N node; 2) any two N nodes can communicate via a route on which no two consecutive nodes are M nodes; 3) among the shared neighbors of two N nodes, at most one third can be compromised. The idea for synchronization in this kind of system is as follows: We trust N nodes to do the computation and information collection. The M nodes are not trustable, but they serve as a channel through which two nearby N nodes exchange their clock readings. Even though this channel is not fully reliable, taking advantage of those one third reliable nodes among the neighbors of any pair of two nearby N nodes impedes the clock reading exchange to be biased due to the malicious nodes.

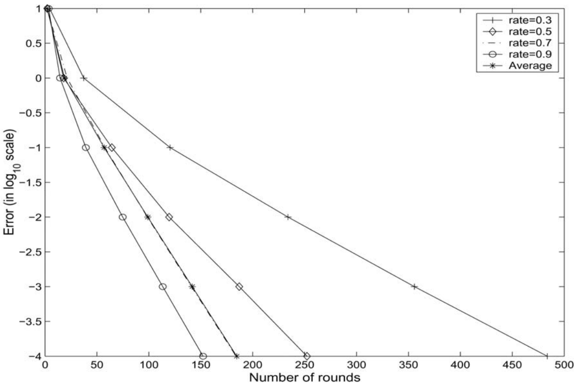


Fig. 7. Convergence speed of average protocol compared with rate-based protocol. The network parameters are: number of nodes 400, sensor field 15x15, transmission range 1.5. We plot the convergence speed for asynchronous rate-based protocol and average protocol. In rate-based protocol, we use rates $0.3, 0.5, 0.7,$ and 0.9 in our simulation. We find the convergence speed increases with a higher rate. The average protocol achieves approximately the same convergence speed as the rate-based for rate equals to 0.7.

The algorithm works in two steps. The first step is clock initialization for M nodes. We ask each N node to broadcast its clock reading to the M nodes since we trust the N nodes but do not trust the M nodes. All its M neighbors should observe the latest received value and set clocks to that value if they are good nodes. In this way, a good M node must have a clock reading that has been corrected by an N node in the initialization phase. In the next step, each N node (say A) collects the clock readings from all its neighbors, averages the clock readings of its neighbors, and broadcasts the average value. The average operation needs a little bit more explanation. Our protocol ensures that each N node (e.g., A) is aware of its neighbors shared with other N nodes (say B). Among those shared neighbors, A chooses the nodes whose values are in the range of the middle one third from the values collected from all neighbors. After A collects the shared node set, it averages the clock readings of all the nodes in the chosen set together with its own value and broadcasts the average value to its neighbors. The neighbors then set their clocks to the newly received value if they are good nodes.

To enforce the correct execution of the operations, we assume the basic cryptographic operations presented in [32], [33]. Our protocol is composed of four basic operations:

1. neighbor discovery,
2. beacon broadcast,
3. the collect operation, and
4. broadcast of the average value.

Neighbor discovery for an N node means finding all the neighbors that are shared with another N node. In beacon broadcast operation, an N node A broadcasts a synchronization message to all its neighbors so that each of its neighbors will record the current clock reading. The collect operation is a composite process in which all the neighbors that receive the broadcast send A their clock readings. In the last step, A broadcasts the average value to all the neighboring nodes and all good neighboring nodes will have a new value after they authenticate the message. We now describe how to implement these operations.

1. Neighbor discovery: Neighbor discovery for an N node is done by broadcasting a neighbor query message and receiving the replies from the neighbors. We claim that a) no neighboring node hides its identity, b) the N node is aware of all its one hop neighbors, and c) two N nodes that are within two hops can communicate their shared neighbors. We discuss how to ensure them as follows: Any node intentionally hiding its identity from an N node does not help in compromising the system since a clock reading from an unidentified node will not be considered by an N node and it cannot impersonate any other M node.

   We eliminate the possibility that a node out of the transmission range becomes a neighbor by considering the packet transmission time. Two hop or multihop nodes will reply to a challenge message with higher latency and, thus, will be eliminated from the one hop neighbor set. Hence, only one hop neighbors are eligible to be in the neighbor set. This precludes nodes that are far away from colluding to conteract the protocol.

   We assume any two nodes share a key by using the Blundo scheme.[7] Therefore, two N nodes within two hops can communicate using the shared key for encryption and via the shared good neighbors. They can thus communicate to each other the set of the shared neighbors.

   We assume that each N node has a special id that can be distinguished by all the nodes. For example, all N node ids could be chosen from the range of [1..100]. An M node that tries to impersonate an N node cannot forge the shared key generated by the Blundo scheme.
2. Beacon broadcast: Broadcast operation uses μTesla [25]. The broadcasting N node first creates a one way key chain $k_1, k_2, \cdots, k_n$, where $k_{i+1} = f(k_i)$ (f is a one way function). It uses the keys in a reverse direction, that is, uses keys in the order of $k_n, k_{n-1}, \cdots, k_2, k_1$. In order to broadcast, the node encrypts the message with the undisclosed key and broadcasts the encrypted message. The next immediate message will be the key used to decrypt the message.[8] A node that fakes an N node's identity will not be able to forge the correctly encrypted message. The neighboring nodes get the message and wait for the key to decrypt the message.
3. Collect operation: The collect operation is done by each node using the shared key (created by the Blundo scheme) with the N node. Other nodes cannot forge the message without knowing the shared key.
4. Average operation: After an N node gets the responses from its neighboring nodes, it computes the average value according to the neighbor sets shared with other N nodes. The node then broadcasts the average value to all its neighbors using μTesla, as in the previous broadcast scheme. Therefore, no one can forge the message except the broadcasting N node. The neighbors then update their clock to the received value.

A malicious node cannot impersonate the identity of other nodes because of the Blundo scheme used. The N node in charge can easily figure out if the claimed node id is authentic by decoding the message encoded by the shared key between the N node and the node with that id (e.g., the message can have the format of (id, current time reading). The N node can see if the id is present in the message after decoding it). We assume that no two nodes far away can collude in this scenario. This can be achieved because each N node would get all its neighbors information in the initial deployment of the sensor network and the node compromise and collusion can only be carried out some time after the deployment since an N node knows those neighbors and it is easy for it to obstruct the message from a node that is not its neighbor.

Theorem 7. *The average algorithm converges to the same value for all the good nodes.*

Proof. Let the highest clock reading of the good nodes be H_t at time t and the lowest clock reading of the good nodes be L_t at time t. Since we use an averaging operation and we always average values that are from good nodes or in the range of $[min_t, max_t]$ (where min_t and max_t are minimal and maximal value of a good node), H_t is nonincreasing over t. Likewise, L_t is nondecreasing over t.

Letting the infimum of the series H_t be H, we have $lim_{t\rightarrow\infty}H_t = H$. Letting the supremum of the series L_t be L, we have $lim_{t\rightarrow\infty}L_t = L$. We have $H_t \geq H$ and $L_t \leq L$ for all t and $H > L$. We want to show that H must be equal to L.

Suppose $H \neq L$. We will derive a contradiction.

Let the function $\epsilon(k) = \sum_{i=1}^{k} n^i\epsilon$. Choose ϵ such that $H - \frac{n^{n+1}-1}{n-1}\epsilon = H - \epsilon(n) = L$, where n is the number of sensors. For any k ($k = n, n-1, \cdots, 1$), define set S_k^1 to be the set of sensors whose values are greater than $H - \epsilon(k)$ and set S_k^2 to be the set of the rest of the nodes.

For ϵ, there must exist a time t such that $H_t < H + \epsilon$; also, at that time, there must be some node whose value is less than $L = H - \epsilon(n)$.

We use the same argument as in Theorem 5. First, look at a snapshot as before. Suppose an N node (A) and all A's good neighbors are in S_k^2 (we can find those nodes considering an N node which just finishes an average operation). Now, consider an N node B that shares neighboring M nodes with A. If B does the average operation, B and all its good neighbors will be in S_{k-1}^2 no matter what value B has before the average operation, so are A and all A's good neighbors because $S_k^2 \subseteq S_{k-1}^2$. Using this as an induction step, we can prove that, eventually, all the good nodes will be in S_1^2. This contradicts that the infimum of H_t is H (i.e., $H_t \geq H$).

[7] The basic idea of the Blundo scheme [2] is presented here succinctly. The details can be found in the original paper. Initially, a special symmetric function $f(x,y) = f(y,x)$ is generated by the key server. A node with id i will be given the function $f(i,y)$. A shared key of node i with another node j will be $f(i,j)$, which can be computed only by nodes i and j.

[8] We assume all the messages can get through without being jammed. And, the key k_n can be sent by the N node to each of its neighbors via a signed message with the shared key generated by the Blundo scheme.

Note that we always average values in the range of $[min_t, max_t]$.

Therefore, H is also within the range of $[min_t, max_t]$. That is, the clocks in the network converge to a reasonable value, instead of any arbitrary one. □

In this protocol, if an N node fails, the M nodes that are only affiliated with this N node will not be able to be synchronized, even though the M node may be a good one.

The assumption that any two neighboring N nodes must share at most one third bad nodes can be justified in the scenario that few nodes are malicious. To some extent, our protocol gives the initial results in this area. Much additional work can be done theoretically to relax the constraints.

This scheme cannot be used in the general cases that assume all the sensor nodes are the same. This method, however, is useful in hierarchical sensor networks, e.g., [24]. In hierarchical networks, special cluster heads are introduced for more powerful processing. If those cluster heads are connected directly within themselves, we can use the diffusion method to synchronize the cluster heads first and then rely on the heads to synchronize all the other nodes.

10 Conclusion

We consider the global synchronization problem in sensor networks. We propose several methods: the all-node-based method, the cluster-based method, the diffusion-based methods, and the fault-tolerant diffusion-based method to solve the problem. The first two methods require a node to initiate the global synchronization, which is neither fault-tolerant nor localized. In the diffusion-based method, each node can perform its operation locally, but still achieve the global clock value over the whole network. We present two implementations of the clock diffusion: synchronous and asynchronous. The synchronous method assumes all the nodes perform their local operations in a set order, while the asynchronous method relaxes the constraint by allowing each node to perform its operation at random. We present the theoretical analysis of these methods and show simulation results for the asynchronous averaging synchronization method. Moreover, we show how to design synchronization protocol in the presence of Byzantine fault.

Our proposed algorithms can be extended to other sensor network applications, such as data aggregation. We are currently examining how the methods presented here fit to more general applications. Our future work also includes implementing the algorithms in a real sensor network using our Mica Mote sensor network platform.

Acknowledgments

The authors thank all the anonynous reviewers for their helpful comments. This project was supported under Award No. 2000-DT-CX-K001 from the Office for Domestic Preparedness, US Department of Homeland Security. Points of view in this document are those of the authors and do not necessarily represent the official position of the US Department of Homeland Security. Support for this work has also been provided in part by MIT's project Intel, Boeing, and US National Science Foundation awards 0423962, EIA-0202789, IIS-0426838, CNS-0520305, CCF-0514985, and IIS-0513755. The authors are grateful for the support.

References

[1] D.P. Bertsekas and J.N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods.* Prentice-Hall, 1989.

[2] C. Blundo, A. Santis, A. Herzberg, S. Kutten, U. Vaccaro, and M. Yung, "Perfectly-Secure Key Distribution for Dynamic Conferences," *Advances in Cryptology (CRYPTO '92),* pp. 471-486, 1993.

[3] J.B. Boillat, "Load Balancing and Poisson Equation in a Graph," *Concurrency: Practice and Experience,* vol. 2, no. 4, pp. 289-313, Dec. 1990.

[4] R.R. Brooks and S.S. Iyengar, "Maximizing Multi-Sensor System Dependability," *Proc. IEEE/SICE/RSJ 1996 Conf. Multisensor Fusion and Integration for Intelligent Systems,* pp. 1-17, Dec. 1996.

[5] R.R. Brooks, N.S.V. Rao, and S.S. Iyengar, "Resolution of Contradictory Sensor Data," *J. Intelligent Automation and Soft Computing,* vol. 3, no. 2, June 1997.

[6] R. Brooks and S.S. Iyengar, "Robust Distributed Computing and Sensing Algorithm," *Computer,* vol. 29, no. 6, pp. 53-60, June 1996.

[7] F. Cristian, "Probabilistic Clock Synchronization," *Distributed Computing,* vol. 3, pp. 146-158, 1989.

[8] G. Cybenko, "Load Balancing for Distributed Memory Multiprocessors," *J. Parallel and Distributed Computing,* vol. 7, no. 2, pp. 279-301, 1989.

[9] D. Dolev, J. Halpern, and H.R. Strong, "On the Possibility and Impossibility of Achieving Clock Synchronization," *Proc. ACM Symp. Theory of Computing (STOC),* May 1984.

[10] J. Elson, L. Girod, and D. Estrin, "Fine-Grained Network Time Synchronization Using Reference Broadcasts," *Proc. Fifth Symp. Operating Systems Design and Implementation (OSDI 2002),* Dec. 2002.

[11] J. Elson and K. Romer, "Wireless Sensor Networks: A New Regime for Time Synchronization," *Proc. First Workshop Hot Topics in Networks (HotNets-I),* Oct. 2002.

[12] L. Girod, V. Bychkovskiy, J. Elson, and D. Estrin, "Locating Tiny Sensors in Time and Space: A Case Study," *Proc. Int'l Conf. Computer Design (ICCD 2002),* Sept. 2002.

[13] J. Halpern, B. Simons, and R. Strong, "Fault-Tolerant Clock Synchronization," *Proc. ACM Symp. Principles of Distributed Computing (PODC),* Aug. 1984.

[14] R.A. Horn and C.A. Johnson, *Matrix Analysis.* Cambridge Univ. Press, 1985.

[15] M. Jerrum and A. Sinclair, "Conductance and the Rapid Mixing Property for Markov Chains: The Approximation of the Permanent Resolved," *Proc. Symp. Theory of Computing,* pp. 235-243, May 1988.

[16] R. Karp, J. Elson, D. Estrin, and S. Shenker, "Optimal and Global Time Synchronization in Sensornets," UCLA technical report, 2003.

[17] B. Krishnamachari and S.S. Iyengar, "Distributed Bayesian Algorithms for Fault-Tolerant Event Region Detection in Wireless Sensor Networks," *IEEE Trans. Computers,* vol. 53, no. 3, Mar. 2004.

[18] L. Lamport and P.M. Melliar-Smith, "Synchronizing Clocks in the Presence of Faults," *J. ACM,* vol. 32, no. 1, pp. 52-78, Jan. 1985.

[19] L. Lamport, "Time, Clocks, and the Ordering of Events in a Distributed System," *Comm. ACM,* vol. 21, no. 7, pp. 558-565, 1978.

[20] C. Liao, M. Martonosi, and D.W. Clark, "Experience with an Adaptive Globally-Synchronizing Clock Algorithm," *Proc. ACM Symp. Parallel Algorithms and Architectures (SPAA),* pp. 106-114, June 1999.

[21] J. Lundelius and N. Lynch, "A New Fault-Tolerant Algorithm for Clock Synchronization," *Proc. ACM Symp. Principles of Distributed Computing (PODC),* pp. 75-88, Aug. 1984.

[22] D.L. Mills, "Internet Time Synchronization: The Network Time Protocol," *Global States and Time in Distributed Systems,* Z. Yang and T. A. Marsland, eds., IEEE CS Press, 1994.

[23] D.L. Mills, "Precision Synchronization of Computer Network Clocks," *ACM/IEEE Trans. Networking,* vol. 6, no. 5, pp. 505-514, Oct. 1998.

[24] J. Pan, Y.T. Hou, L. Cai, Y. Shi, and S.X. Shen, "Topology Control for Wireless Sensor Networks," *Proc. ACM Mobicom,* pp. 286-299, 2003.

[25] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J.D. Tygar, "SPINS: Security Protocols for Sensor Networks," *Proc. Seventh Ann. Int'l Conf. Mobile Computing and Networks (MobiCOM 2001),* July 2001.

[26] P. Ramanathan, K.G. Shin, and R.W. Butler, "Fault-Tolerant Clock Synchronization in Distributed Systems," *Computer,* pp. 33-42, Oct. 1990.

[27] K. Romer, "Time Synchronization in Ad Hoc Networks," *Proc. ACM MobiHoc,* Oct. 2001.

[28] W. Su and I. Akyildiz, "Time-Diffusion Synchronization Protocol for Sensor Networks," *IEEE/ACM Trans. Networking,* Feb. 2005.

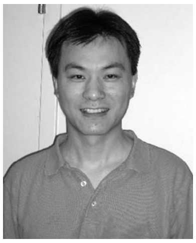
[29] R. Subramanian and I.D. Scherson, "An Analysis of Diffusive Load-Balancing," *Proc. Sixth Ann. ACM Symp. Parallel Algorithms and Architectures,* pp. 220-225, 1994.

[30] C.-Z. Xu and F.C. M. Lau, "Analysis of the Generalized Dimension Exchange Method for Dynamic Load Balancing," *J. Parallel and Distributed Computing,* vol. 16, no. 4, pp. 385-393, Dec. 1992.

[31] C. Xu and F.C. M. Lau, *Load Balancing in Parallel Computers: Theory and Practice.* Kluwer Academic, 1997.

[32] S. Zhu, S. Setia, and S. Jajodia, "Leap: Efficient Security Mechanisms for Large-Scale Distributed Sensor Networks," *Proc. 10th ACM Conf. Computer and Comm. Security,* Oct. 2003.

[33] S. Zhu, S. Setia, S. Jajodia, and P. Ning, "An Interleaved Hop-by-Hop Authentication Scheme for Filtering of Injected False Data in Sensor Networks," *Proc. IEEE Symp. Security and Privacy,* May 2004.



Qun Li received the PhD degree in computer science from Dartmouth College. He is an assistant professor in the Department of Computer Science at the College of William and Mary. His research interests include wireless networks and sensor networks. He is a member of the IEEE.

Daniela Rus received the PhD degree in computer science from Cornell University. is an associate professor in the Electrical Engineering and Computer Science Department at the Massachusetts Institute of Technology. Previously, she was a professor in the Computer Science Department at Dartmouth College. Her research interests include distributed robotics, mobile computing, and self-organization. She was the recipient of a US National Science Foundation Career award. She is an Alfred P. Sloan Foundation Fellow and a class of 2002 MacArthur Fellow. She is a member of the IEEE.