Global Clock Synchronization in Sensor Networks

Qun Li[*]
[*]Department of Computer Science
Dartmouth College
Hanover, NH 03755
Email: liqun@cs.dartmouth.edu

Daniela Rus[*][†]
[†]CS and AI Lab
MIT
Cambridge, MA 02139
Email: rus@cs.dartmouth.edu

Abstract— **Global synchronization is crucial to many sensor network applications that require precise mapping of the 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 discusses three methods to achieve global synchronization in a sensor network: a node-based approach, a hierarchical cluster-based method, and a fully localized diffusion-based method. We also give the synchronous and asynchronous implementations of the diffusion-based protocols.**

I. Introduction

Many emerging sensor network applications require that the sensors in the network agree on the time. A global clock in a sensor system will help process and analyze the data correctly and predict future system behavior. For example, in the 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. Other applications that need global clock synchronization include environment monitoring (for example, temperature), navigation guidance, and any other application that requires the coordination of locally sensed data and mobility. Clock synchronization may also help to conserve energy in a sensor network, by allowing a coordinated way to set nodes into sleeping mode. This leads to more complex communication since a node must compute when to wake up to receive a message.

In this paper we discuss three methods for global synchronization in a sensor network: (1) the all-node-based method, (2) the cluster-based method, and (3) a fully localized diffusion-based method. The all-node-based method assumes the transmission time of a packet across a hop is the same for all nodes. It uses a packet to go around a cycle that is composed of all the nodes in the network and amortizes the packet transmission time on the cycle to each hop. This method does not scale well because it requires the nodes in the whole network to participate in the synchronization process at the same time. To address the scalability issue, we propose a hierarchical method. We use clusters to organize the whole network. The cluster head nodes are synchronized by using the first method and in each cluster the members are synchronized with the cluster head. These two methods are not localized; each synchronization process involves all the nodes in that network partition. To achieve full scalability, we propose 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 proportional to their clock difference in a set order. To make our implementation more practical, we propose two asynchronous implementations, in which a node can synchronize with its neighbors at any time in any order. The asynchronous algorithms can also adapt to node failure, adverse communication channel, and node mobility.

Although our algorithms are aiming at solving the synchronization problem in a sensor network, they can be easily extended to the data aggregation problem, e.g., finding the average, highest, and lowest sensor data reading among all the sensors in the whole network. For example, in the all-node-based algorithm, the reading of a sensor can be attached to the message when the message used in the all-node-based synchronization is going along the cycle composed of all nodes. 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. The diffusion-based algorithm can also be extended straightforwardly.

This paper is organized as follows. Section III presents the overview of the problem. Section IV discusses a synchronization scheme that requires all the nodes to participate in the global synchronization when a node initiates a synchronization request. Section V describes a cluster-based scheme that reduces the number of the participating nodes. Section VI gives a synchronous diffusion-based algorithm that is fully localized. Section VII discusses two asynchronous diffusion-based algorithms. Section VIII gives the simulation results on the asynchronous averaging algorithm.

II. Related Work

Synchronization has been studied for a long time in traditional computer and embedded systems [13], [14], [12], [3], [15]. The classical paper on logical time [11] presented the solution to causal ordering of events in a distributed system. Papers on synchronization in sensor networks include [6], [5], [7], [16].

Cristian [3] proposed a probabilistic synchronization method that exploits a large number of messages to get the accurate shortest round-trip time with high probability. Ramanathan et al. [15] surveyed fault-tolerant clock synchronization methods in distributed systems. Romer [16] used message delay, which is estimated by the lower bound 0 and the upper bound round trip time, to compare the maximal difference between two communicating nodes and thus synchronize them. The solution targets an ad hoc network in which two nodes may be out of range and lose their direct communication by propagating the maximal estimated clock difference along the communication path. Elson and Romer [6] discussed the design principles for synchronization in sensor networks: use multiple, tunable modes of synchronization; avoid maintain a global timescale for the entire network; use post-facto synchronization; adapt to application; and exploit domain knowledge. Elson et al. [5] proposed 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 multi-hop scenario. The three papers provided important theoretical and practical building blocks for sensor network synchronization. However, they did not specify how to do global synchronization over the entire network.

In Intanagonwiwat et al.'s direct diffusion [9] approach, data generated by sensor nodes is named by attribute-value pairs. A node requests data by sending interests for named data; the interests are propagated within the network to find the source of the related data. The direct diffusion method is used to reinforce the best path from the source to the sink. The goal of our diffusion-based algorithm is to use local operations to achieve global consensus.

Diffusion methods have been used in load balancing [4], [2], [18], [1], [17]. 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 under-loaded. Cybenko [4] and Boillat [2] analyzed the diffusion method in load balancing. They gave the sufficient and necessary condition for the convergence of the method. The time complexity of the diffusion method was analyzed in [17]. Initial result about the asynchronous diffusion method is provided in [1], which gave the convergence proof of the asynchronous rate-based protocol, but not the average protocol. We think it is helpful that we put our different proof here for completeness. These load balance algorithms fit into the robust interconnection network well. However, further investigation is needed to see how they can be applied to sensor network applications in which the communication channel is not perfect, nodes are prone to failure, and the system may be mobile.

III. 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 $C(t)$ leads to the events (or interrupts) that can be captured by the sensor.

The clocks in a sensor network can be inconsistent due to several reasons. 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. The nodes in a sensor network may not be synchronized well initially, when the network is deployed. The sensors may be turned on at the different times and their clocks may be running according to different initial values. The results of events on specific sensors may also affect the clock. For example, the Berkeley 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.

We first explore how well the synchronization precision can be achieved in an ideal system conforming to our assumptions.

Theorem 1: Ideally, it is possible to bound the clock difference in a sensor network to Δ (that is, any two clocks differ by at most Δ), but not less than $\Delta/2$, where Δ is the clock cycle in use.

Proof: We can use a global wall clock to synchronize all the clocks in the network. At the wall clock time t, each clock adjusts to set its next clock interruption point to t. For any node the clock cannot be earlier than the wall clock, but is later by at most Δ, one clock cycle time. Thus the error between any two node clocks is at most Δ, the clock cycle. On the other hand, there is no way that we are able to set their difference by less than $\Delta/2$. An example would be the clock of one node ticks $\Delta/2$ time later than that of another node. Since we cannot change when a clock should tick, their clock difference is at least $\Delta/2$. ■

[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.

IV. All-node-based Synchronization

In this section we describe a method to globally synchronize the clocks in a sensor network. We assume the clock cycle on each node is the same. This is reasonable since in most of the cases before the sensors are deployed they are programmed with the same parameters. We also assume the clock tick time is much longer than the packet transmission time[2].

Our algorithm assumes the message transmission time and handling time on each node is roughly the same. This can be obtained when the network traffic is small, e.g., upon its initial deployment, a sensor network allows sufficient time solely for clock synchronization (during the synchronization time, all other messages except the synchronization messages are suppressed for transmission). The main idea of the algorithm 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. Alg. 1 summarizes this method.

Alg. 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 cycle. 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 has traveled so far. In the second phase, a clock correction packet is sent along the same cycle informing each node the packet starting time, ending time at the initiating node and the total hops in the cycle. Each node then computes how to adjust its clock.

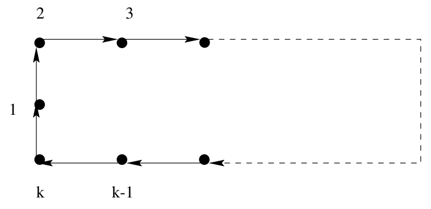

Fig. 1. A clock synchronization message traveling along a loop originated from node n_1 and then back to node n_1.

Consider the example in Fig. 1. Node n_1 initiates the clock synchronization and generates a message with its current time, t_s, attached. The time t_s is the exact tick time of 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 sends it out in the same way as the previous message. The message will travel along the same path; this time each sensor will try to get the hops this sensor is away from node n_1 (include 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 Alg. 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 line 7 and 9, a node adjusts its clock using different value of m obtained from line 6 and 8 respectively.

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

Algorithm 1 All Node based Synchronization Algorithm in a Sensor Network

```
1: Find a cycle that passes each node at least once that need to be synchronized
2: A message is passed along the cycle starting from an initiating node
3: Upon receipt of the message, each node records its current local time ($t_i$) and its order ($i$) in the cycle. If the node receives messages more than once, it chooses one arbitrarily.
4: After the initiating node receives the message, it sends out another message informing each node on the cycle the start time ($t_s$) and the end time ($t_e$) of the previous message
5: for each node, to adjust its local time $t$ do
6:   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
7:     node $n_i$ adjusts its time to $t - t_i + t_s + m$
8:   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
9:     node $n_i$ adjusts its time to $t - t_i + t_s + m$
```

Theorem 2: After running Alg. 1 the relative clock error between any two nodes is at most 3Δ.

Proof: Without loss of generality, the proof in the following looks at 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 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.

Algorithm 2 The Cluster Synchronization Algorithm

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

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.

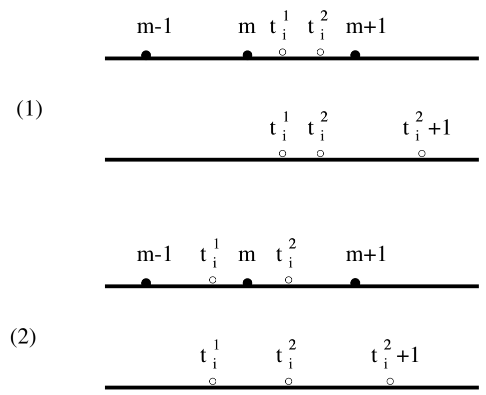

Fig. 2. n_i receives the synchronization message at time t_i such that $t_i^1 \leq t_i \leq t_i^2$. Since $|t_i^2 - t_i^1| < 1$, t_i^1 and t_i^2 are either in the range of two consecutive integers or separated by only one integer. Those two figures describe the two cases. On the second line in each figure, 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 $m+1$.

For the first case, the error (the value obtained by subtracting n_1 time from n_i 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 between $t_s + m - \frac{i-1}{k}$ and $t_s + m + \frac{i-1}{k} + 1$, so the maximal error is $(t_s + m + 1 - (t_s + m - \frac{i-1}{k}))\Delta = (1 + \frac{i-1}{k})\Delta$. 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 error (the value obtained by subtracting n_1 time from the n_i time) is 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 $-(1+\frac{i-1}{k})\Delta$, another one has 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).

V. CLUSTER-BASED SYNCHRONIZATION

The synchronization method proposed in Section IV has a provable bound but 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 Alg. 2. In Alg. 2, we first use the same method as in Alg. 1 to synchronize all the cluster heads by designing a message path that contains all the cluster heads (we call the initiators base). Then in the second step, the nodes in each cluster can be synchronized with their 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 the nodes in a large area (e.g., a zone). For the first case, RBS can be used. First a reference broadcast is sent out by the head to synchronize all the other cluster members, then any other node in the cluster sends out another reference broadcast to synchronize all the head and the other members. By these two broadcasts, the clock difference can be calculated, and all the non-head members can adjust their clocks according to the head's clock. In a zone scheme, we can use the same method as Alg. 1 to first design a cycle to include all the nodes of the cluster and synchronize them all. The head of the cluster will be the initiator of the intra-cluster synchronization.

Using Alg. 1 in both the two hierarchical levels 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 inter-cluster synchronization path of length k and j is the order of node n_{ij} on the intra-cluster 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.

VI. SYNCHRONOUS DIFFUSION

A. 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 develop a diffusion method that is fully distributed and localized. The synchronization is done only locally, without a global synchronization initiator. It can also be done at a relaxed time constraint as opposed to the stringent requirement of the previous methods.

Our diffusion method achieves global synchronization by spreading the local synchronization information to the whole system. It can choose various global values to synchronize the network provided that each node in the overall network agrees to change its clock reading to the consensus value. An easy and possible way is to choose the highest or lowest reading over the network. We will show that synchronization to the highest or lowest value entails a simple algorithm. However, a faulty or malicious node may impose an abnormally high or low clock reading, which is likely to mess up 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.

Our discussion is based on the two basic operations: (1) the neighboring nodes compare their clock readings at a certain time point and (2) change them 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.

Our goal here is to present 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[5] 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 neighbor (say B) sends out another reference broadcast, so 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 neighbor. In another example that assumes 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 $t_2 - t_1 - t_0$.

We argue that in some real time applications, such as vehicle tracking, post-facto synchronization is not sufficient. We aim at synchronizing all the clocks of the network in real time. RBS can also achieve global synchronization, but it is not a localized protocol. Our diffusion-based protocols are localized holding the value of adapting to node failure, broken link, and node mobility.

B. 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 that graph, and if two sensors are within the transmission range of each other, their corresponding vertices n_i and n_j have an edge to connect them.

Algorithm 3 Diffusion algorithm to synchronize the whole network

```
1: Do the following with some given frequency
2: for each sensor $n_i$ in the network do
3:     Exchange clock times with $n_i$'s neighbors
4:     for each neighbor $n_j$ do
5:         Let the time difference between $n_i$ and $n_j$ be $t_i - t_j$
6:         Change $n_i$'s time to $t_i - r_{ij}(t_i - t_j)$
```

Let $C = (c_1^t, c_2^t, \cdots, c_n^t)^T$ 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 $c_i - \sum_{j \neq i} r_{i,j} \cdot (c_i - c_j) = (1 - \sum_{j \neq i} r_{i,j}) \cdot c_i + \sum_{j \neq i} r_{i,j} \cdot c_j$

Alg. 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.

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 Alg. 3, we apply the 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.

C. Convergence of the Rate-based Synchronous Algorithm

In this section we show that Alg. 3 achieves global synchronization in the whole 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 similar to the convergence of a Markov chain and then present results regarding the convergence speed.

We assume the graph we derived from the network is strongly connected, so the matrix R is irreducible. It is also symmetric and positive because $r_{ij} = r_{ji} > 0$. The eigenvalues of a symmetric matrix are real. Also by Perron-Frobenius theorem [8], the matrix R has the following eigenvalues: $1 = \lambda_1 > \lambda_2 \geq \lambda_3 \geq \cdots \geq \lambda_n \geq -1$.

Theorem 3: The convergence of Alg. 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 $\sum_{i=1}^{n} a_i e_i$, then 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 4: If $\lambda_n = -1$, then there is a permutation matrix P such that $P^T RP$ has the 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 (remember the eigenvalues of a symmetric matrix are always real.), and there is a permutation matrix P such that $P^T RP$ has the 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 $r_{ii} \neq 0$, our 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 5: For R with eigenvalue $\lambda_{max} < 1$ and synchronized clock vector C^s, the relative error after running Alg. 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 $c_i^t = \sum_{j=1}^{n} r_{ij}^{(t)} \cdot c_j^0 = \frac{1}{n} \sum c_j^0 + \sum (r_{ij}^{(t)} - \frac{1}{n}) c_j^0 = c^0 + \sum (r_{ij}^{(t)} - \frac{1}{n}) c_j^0$. Thus, the relative error $\frac{|c_i^t - c^0|}{c^0} = \frac{\sum (|r_{ij}^{(t)} - \frac{1}{n}|) c_j^0}{c^0} \leq n\lambda_{max}^t$ (since $c_j^0 \leq 1$). ■

Algorithm 4 Asynchronous Averaging Algorithm in a Sensor Network

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

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 $\Phi = min_{S: C_S \leq \frac{1}{2}} \Phi_S$. By Jerrum-Sinclair [10], $\lambda_2 < 1 - \Phi^2/2$. Thus, we can bound the convergence speed if we know Φ.

VII. The Asynchronous Diffusion Methods

A. Asynchronous Diffusion Algorithms

In the previous section we gave the synchronous version of the rate-based algorithm, proved its convergence, and bound its convergence speed. It is nice that the synchronous algorithm is localized. However, it requires the node operations to be done in a set order. No node can perform the operation without waiting for all the nodes to finish the current round of operations. To make our algorithm more practical, we present an asynchronous version of the algorithm that does not have 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 non-zero probability. Our method can be shown to converge.

We start with an asynchronous averaging algorithm (Alg. 4), which is mainly 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 to update their values.

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. As before, we also assume the network is connected.

Theorem 6: 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 non-increasing 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 non-decreasing over time t by symmetry.

We know $H_t \geq C$ and H_t is non-increasing according to (1). Let the infimum of the series H_t be M, we have $lim_{t \to \infty} H_t = M \geq C$. Suppose $M \neq C$. We will derive a contradiction.

Define 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 set S_k^1 to be the set of sensors whose values are greater than $M - \epsilon(k)$ 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 $H_t < M + \epsilon$; 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 k-th 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 $|S_n^2| \geq 1$ (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 $|S_{n-1}^2| \geq 2$. After the first average operation on nodes in S_{n-1}^1 and S_{n-1}^2 we have $|S_{n-2}^2| \geq 3$. So eventually, we have $S_1^2 = n$ [3]. This contradicts that the infimum of H_t is M (i.e., $H_t \geq M$).

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 7: The asynchronous rate-based algorithm converges to the global average value.

Proof: We prove this result using an approach similar to Theorem 6. Let the $min\ r_{ij} = r$ ($i \neq j$ 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: $M + \epsilon - r_{ij}(M + \epsilon - (M - \epsilon(k))) \leq M + \epsilon - r(M + \epsilon - (M - \epsilon(k))) = M + \epsilon - r(\epsilon + \epsilon(k)) = M + \epsilon - r(\epsilon + \frac{\epsilon}{r^{n+1-k}} - \epsilon) = M + \epsilon - \frac{\epsilon}{r^{n-k}} = M - \epsilon(k-1)$. In this way, we can prove the asynchronous rate-based algorithm converges. ■

We 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 $n_i \in V$ and $n_j \in V$) if and only if n_i and n_j involve in the same average operation[4] after time t. The sensor network is operation connected if $G_t(V, E)$ is connected for any $t > 0$. Alg. 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 non-zero probability to be 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, (3) the sensor nodes are mobile.

Now we discuss the convergence speed of the asynchronous algorithms. As before, we define the error to be $\frac{H_t - C}{C}$ 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 non-zero values. We can think of this case as n copies of networks 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 the 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 only need to consider a network with only one node that has non-zero 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 actually is corroborated 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.

B. Discussion

Pure RBS method can be used to compute the clock difference between any two nodes by subsequently comparing the nodes on a path connecting the two nodes. By [5], the receive error, which translates to the clock difference in a RBS operation, is Gaussian distributed with variance σ. For two nodes with k hops apart, the average clock difference is $\sqrt{k}\sigma$. Note that here we ignore the clock drift in this multi-hop RBS process because the drift is very small in the short period of multi-hop communication.

In our diffusion-based scheme (suppose we use RBS as our local synchronization method), the clock difference between any two nodes is comprised of three types of errors: (1) the receive error as in the generic RBS scheme, (2) the clock drift of any node after the last RBS operation, (3) the error due to the limited diffusion speed in our diffusion scheme (let it be ϵ_0). The clock drift is negligible due to the same reason as in the generic RBS scheme. The receive error and the diffusion

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

[4]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.

error are independent. The receive error did not appear in the diffusion error in our previous analysis as we assumed that all the nodes participating in the same RBS operation are synchronized perfectly. Thus, the clock difference between any two nodes in the network is the sum of the receive error and the diffusion error, which is $\sqrt{k}\sigma + \epsilon_0$ where k is the number of hops that separate the two nodes apart. The above analysis assumes the time can be represented continuously on each node, that is, any clock has perfect resolution on time. This is true if the tick time Δ is small. When Δ is predominant, the precision of diffusion-based method is subject to the tick time as the first two protocols. The errors in our diffusion-based protocols are $k\Delta$ where k is the maximal number of hops that separate any two nodes in the network apart.

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. For example, we can reduce the clock error by using multiple RBS operations. The idea is to execute repeatedly the RBS operations to reduce the clock difference variance. Suppose two nodes participate in m RBS operations in a short period, we can get m copies of their clock difference, $d_1, d_2, \cdots, d_m$, which are all Gaussian distributed with average d and deviation σ Thus, $\frac{\sum_{i=1}^{m} d_i}{m}$ is Gaussian distributed with average d and deviation $\frac{\sigma}{\sqrt{m}}$.

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 of the applications events are processed locally. For example, in that vehicle tracking application, the sensors that sense 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 the 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.

VIII. Simulation

We implemented the averaging synchronization algorithm (Alg. 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 Alg. 4 exactly once, so the number of rounds for the network to achieve some error threshold signifies the convergence speed.

For each experimental set with the same parameters, the simulation was executed several times using a randomly generated network topology. In each experiment, a stimulus is generated at a randomly chosen node and propagated to the whole network until the relative error is achieved. In Fig. 6, 3, and 4, 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:

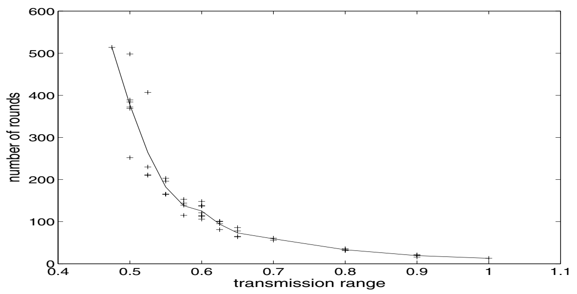

Fig. 3. The convergence speed with different transmission ranges. The network parameters are: sensor field 10x10, number of nodes 1000, algorithm stops at error 0.01%. The crosses signify the number of rounds for each experiment. The plotted curve is the average number of rounds for networks with the same network parameters.

1) The first suite of experiments (Fig. 5) relate the relative error with the number of rounds executed by the sensors. We generated two random networks with the following network parameters: number of nodes 200, sensor field 10x10, transmission range 1.5, and algorithm stop error 0.01%. As we conjecture in the previous section, the

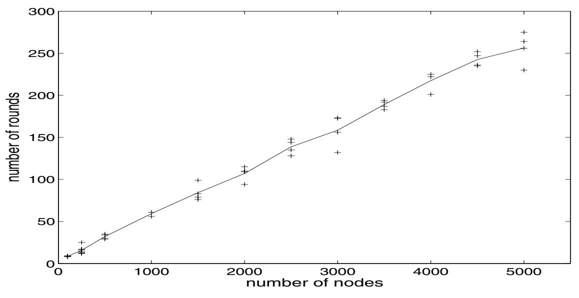

Fig. 4. The convergence speed with the number of nodes with fixed node density. The network parameters are: sensor field proportional to the number of nodes (10 nodes per unit area), transmission range 0.7, algorithm stops at error 0.1%. We see almost linear decrease in the convergence speed with the increase of the nodes. The plotted curve is the average number of rounds for networks with the same network parameters.

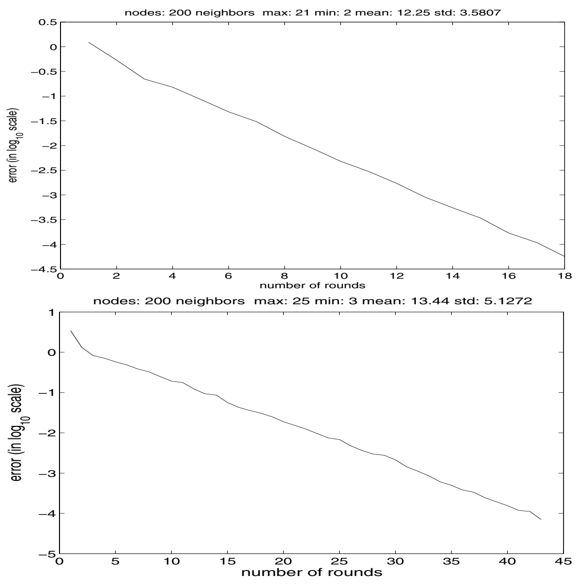

Fig. 5. Two typical simulations for average algorithm. The experiments have the following parameters: number of nodes 200, sensor field 10x10, transmission range 1.5, and algorithm stop error 0.01%. The number of neighbors of each node varies in each experiment. In the first experiment, the maximal number 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.

error rate decreases exponentially with the increase of the number of rounds. The simulation confirms our conjecture.

2) Convergence speed vs. number of nodes with other parameters fixed (Fig. 6). 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 with the following parameters: sensor field 10x10, transmission range 1.5, and algorithm stop error 0.01%. 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

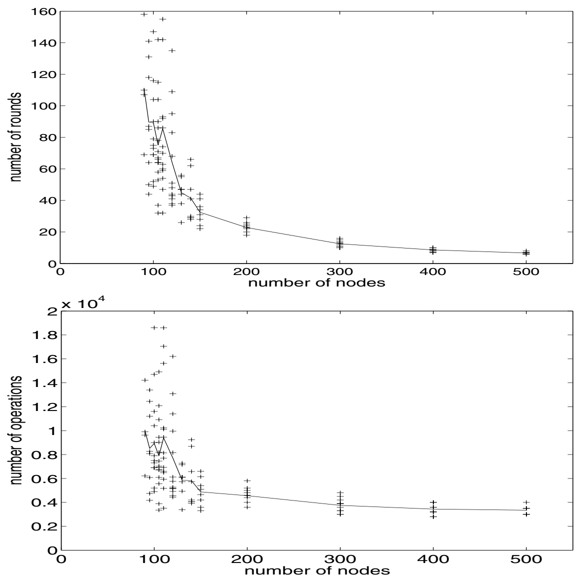

Fig. 6. The convergence speed with different number of nodes. The network parameters are: sensor field 10x10, transmission range 1.5, and algorithm stop error 0.01%. The markers are the number of rounds for each experiment. The plotted curve is the average number of rounds for one suite of network parameters. A sparse network with less 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.

total operations for one suite of network parameters. A sparse network with less 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 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 vs. various network setups (Fig. 7). We did simulations on four suites of network setups. In each setup, we generated multiple random networks with the same network parameters. The first figure uses the following parameters: number of nodes 200, sensor field 10x10, transmission range 1.5. The second figure uses: number of nodes 400, sensor field 10x10, transmission range 1.5. The third figure uses: number of nodes 400, sensor field 15x15, transmission range 1.5. The fourth figure uses: number of nodes 400, sensor field 18x18, transmission range 1.5. In each sub-figure, the lines represent the performance variations due to the randomly generated network topologies. With the same network scope and transmission range, the convergence speed increases with the increase of the number of nodes (by comparing the first and second figures). The reason is that when the number of neighbors of each node increases, the network is more connected, which

expedites the diffusion. With the same number of nodes and transmission range, the average number of neighbors of a node decreases with the increase of the network scope. This leads to a less connected network, then the convergence speed decreases (by comparing the second, third, and fourth figures).

4) Convergence speed vs. transmission range with other parameters fixed (Fig. 3). The figure evaluates the convergence speed with different transmission ranges. The network parameters are: sensor field 10x10, number of nodes 1000, algorithm stops at error 0.01%. The crosses signify the number of rounds for each experiment. 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 decreases with the increase of the transmission range. This is due to the fact that the number of neighbors of each node increase and the number of nodes covered by each average operation increases.
5) Convergence speed vs. number of nodes with fixed sensor density (Fig. 4). 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%. 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 in each simulation, the diffusion speed is dependent on the number of sensors in the whole network.

IX. Conclusion

We consider the global synchronization problem in sensor networks. We propose the all-node-based method, the cluster-based method, and the diffusion-based methods 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 constrain 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.

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.

Acknowledgment

We thank all the reviewers for their helpful comments.

This project was supported under Award No. 2000-DT-CX-K001 from the Office for Domestic Preparedness, U.S. 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 U.S. Department of Homeland Security. Support for this work has also been provided in part by the Institute for Security Technology Studies (ISTS), MIT project Oxygen, NSF award 0225446, I1S-9912193, ONR award N00014-01-1-0675, and DARPA TASK program award F30602-00-2-0585.

References

[1] D.P. Bertsekas and J.N. Tsitsiklis. *Parallel and distributed computation: Numerical methods*. Prentice-Hall Inc., 1989.

[2] J.B. Boillat. Load balancing and poisson equation in a graph. *Concurrency: Practice and Experience*, 2(4):289–313, December 1990.

[3] Flaviu Cristian. Probabilistic clock synchronization. *Distributed Computing*, 3:146–158, 1989.

[4] George Cybenko. Load balancing for distributed memory multiprocessors. *Journal of Parallel and Distributed Computing*, 7(2):279–301, 1989.

[5] Jeremy Elson, Lewis Girod, and Deborah Estrin. Fine-grained network time synchronization using reference broadcasts. In *ACM OSDI 2002*, Boston, MA, December 2002.

[6] Jeremy Elson and Kay Rmer. Wireless sensor networks: A new regime for time synchronization. In *First Workshop on Hot Topics In Networks (HotNets-I)*, Princeton, New Jersey, October 2002.

[7] Lewis Girod, Vladimir Bychkovskiy, Jeremy Elson, and Deborah Estrin. Locating tiny sensors in time and space: A case study. In *International Conference on Computer Design (ICCD 2002)*, Freiburg, Germany, September 2002.

[8] Roger A. Horn and Charles A. Johnson. *Matrix Analysis*. Cambridge University Press, 1985.

[9] Chalermek Intanagonwiwat, Ramesh Govindan, and Deborah Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *MobiCOM 2000*, Boston, Massachusetts, August 2000.

[10] M. Jerrum and A.Sinclair. Conductance and the rapid mixing property for markov chains: the approximation of the permanent resolved. In *Symposium on Theory of Computing*, pages 235–243, May 1988.

[11] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978.

[12] Cheng Liao, Margaret Martonosi, and Douglas W. Clark. Experience with an adaptive globally-synchronizing clock algorithm. In *ACM SPAA*, pages 106–114, New York, June 1999.

[13] David L. Mills. Internet time synchronization: the network time protocol. In Zhonghua Yang and T. Anthony Marsland, editors, *Global states and time in distributed systems*. IEEE Computer Society Press, 1994.

[14] David L. Mills. Precision synchronization of computer network clocks. *ACM/IEEE Transactions on Networking*, 6(5):505–514, Oct. 1998.

[15] Parameswaran Ramanathan, Kang G. Shin, and Ricky W. Butler. Fault-tolerant clock synchronization in distributed systems. *IEEE Computer*, pages 33–42, Oct. 1990.

[16] Kay Romer. Time synchronization in ad hoc networks. In *ACM MobiHoc*, Long Beach, CA, Oct. 2001.

[17] R. Subramanian and I.D. Scherson. An analysis of diffusive load-balancing. In *ACM SPAA*, pages 220–225, 1994.

[18] C.-Z. Xu and F.C.M. Lau. Analysis of the generalized dimension exchange method for dynamic load balancing. *Journal of Parallel and Distributed Computing*, 16(4):385–393, December 1992.

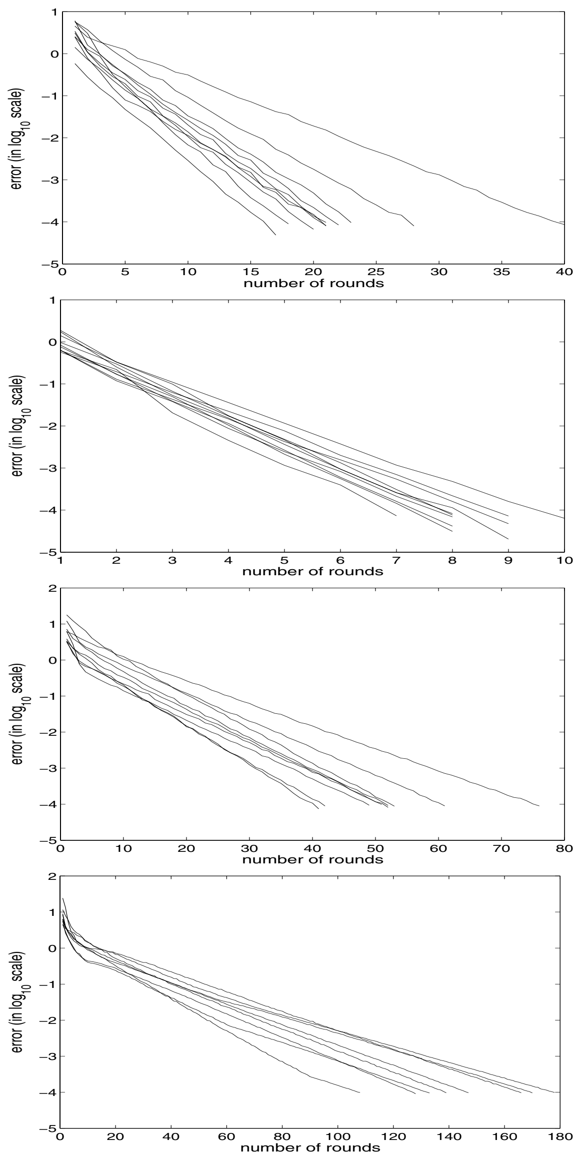

Fig. 7. Simulation results of different experimental sets. In each figure, we fix the parameters of the network, but obtain different networks by randomly generating the node positions (the different lines in each figure represent different networks generated). For each network experimental set, we run the algorithm till the error is below 0.01% (error is defined as $\frac{H_t - C}{C}$ where H_t is the highest value at that time and C is the average value in the end). In each round, we randomly generate the node execution order although we force each node to execute the average operation once in each round. We vary the number of nodes and the network scope for each experimental set. The first figure uses the following parameters: number of nodes 200, sensor field 10x10, transmission range 1.5. The second figure uses: number of nodes 400, sensor field 10x10, transmission range 1.5. The third figure uses: number of nodes 400, sensor field 15x15, transmission range 1.5. The fourth figure uses: number of nodes 400, sensor field 18x18, transmission range 1.5. This verifies our conjecture that the relative error decreases exponentially with the increase of the number of rounds. We can see some effects on the convergence speed of the network scope, number of nodes and transmission range from these figures.