Timing-sync Protocol for Sensor Networks

Saurabh Ganeriwal   Ram Kumar   Mani B. Srivastava

Networked and Embedded Systems Lab (NESL), University of California Los Angeles

56-125B Eng. IV, UCLA EE Dept., Los Angeles, CA 90095

{saurabh, ram, mbs}@ee.ucla.edu

ABSTRACT

Wireless ad-hoc sensor networks have emerged as an interesting and important research area in the last few years. The applications envisioned for such networks require collaborative execution of a distributed task amongst a large set of sensor nodes. This is realized by exchanging messages that are time-stamped using the local clocks on the nodes. Therefore, time synchronization becomes an indispensable piece of infrastructure in such systems. For years, protocols such as NTP have kept the clocks of networked systems in perfect synchrony. However, this new class of networks has a large density of nodes and very limited energy resource at every node; this leads to scalability requirements while limiting the resources that can be used to achieve them. A new approach to time synchronization is needed for sensor networks.

In this paper, we present Timing-sync Protocol for Sensor Networks (TPSN) that aims at providing network-wide time synchronization in a sensor network. The algorithm works in two steps. In the first step, a hierarchical structure is established in the network and then a pair wise synchronization is performed along the edges of this structure to establish a global timescale throughout the network. Eventually all nodes in the network synchronize their clocks to a reference node. We implement our algorithm on Berkeley motes and show that it can synchronize a pair of neighboring motes to an average accuracy of less than 20µs. We argue that TPSN roughly gives a 2x better performance as compared to Reference Broadcast Synchronization (RBS) and verify this by implementing RBS on motes. We also show the performance of TPSN over small multihop networks of motes and use simulations to verify its accuracy over large-scale networks. We show that the synchronization accuracy does not degrade significantly with the increase in number of nodes being deployed, making TPSN completely scalable.

Categories and Subject Descriptors

C.2.2 [**Computer Systems Organization**]: Computer Communication Networks – *Network Protocols.*

General Terms

Algorithms, Experimentation, Performance, Verification.

Keywords

Sensor Networks, Time Synchronization, Packet Delay, Clock Drift, Medium Access Control.

SenSys '03, November 5-7, 2003, Los Angeles, California, USA.

1. INTRODUCTION

Advances in microelectronics fabrication have allowed the integration of sensing, processing and wireless communication capabilities into low-cost and small form-factor embedded systems called sensor nodes [1], [2]. The need for unobtrusive and remote monitoring is the main motivation for deploying a sensing and communication network (sensor network) consisting of a large number of these battery-powered nodes.

The applications envisioned for sensor networks vary from monitoring inhospitable habitats and disaster areas to operating indoors for intrusion detection and equipment monitoring. Most of these applications require sensor nodes to maintain local clocks in order to determine the timing of the events. In general, sensor nodes gather sensor readings, and use several signal processing techniques to get meaningful results from this raw data. For example, the target tracking applications use Kalman filter to estimate the target position [3]. Such signal-processing techniques require relative synchronization among sensor node clocks, so that a right chronology of events can be detected. On the other hand, for sensor network applications such as detecting brushfires, gas leaks etc.; the time of occurrence of an event is itself a critical parameter. For such class of applications, the synchronization of the complete network with every node maintaining a unique global time scale becomes paramount. Time synchronization is also indispensable in the implementation of the commonly used medium access control (MAC) protocols such as TDMA [4].

Time synchronization problem has been investigated thoroughly in Internet and LANs. Several technologies such as GPS, radio ranging etc have been used to provide global synchronization in networks. Complex protocols such as NTP [5] have been developed that have kept the Internet's clocks ticking in phase. However, the time synchronization requirements differ drastically in the context of sensor networks. In general such networks are dense, consisting of a large number of sensor nodes. To operate in such large network densities, we need the time synchronization algorithm to be *scalable* with the number of nodes being deployed. Also, *energy efficiency* is a major concern in these networks due to the limited battery capacity of the sensor nodes. This eliminates the use of external energy-hungry equipments, such as GPS. Moreover, time synchronization requirements are much more stringent, often requiring synchronization of the order of microseconds among nodes involved in a task such as tracking a target.

1.1 Contributions

We present a Timing-sync Protocol for Sensor Networks (TPSN) that works on the conventional approach of sender-receiver synchronization. We argue that for sensor networks, the classical approach of doing a handshake between a pair of nodes is a better approach than synchronizing a set of receivers [6]. This observation comes as result of time stamping the packets at the moment when they are sent *i.e.,* at MAC layer, which is indeed feasible for these networks. To prove our claim, we compare the performance of TPSN with Reference Broadcast Synchronization (RBS) [7], a timing synchronization algorithm for sensor networks based on receiver-receiver synchronization. We will show that TPSN gives roughly a 2x better performance than RBS via implementation on motes. We will show that TPSN can synchronize a pair of motes to an average accuracy of less than 20μs and a worst-case accuracy of around 50μs.

In sensor networks, clock synchronization might not be needed all the times. For such scenarios, TPSN can be combined with the approach of post-facto synchronization proposed in [8] to provide time synchronization among a subset of nodes. Post-facto synchronization is used to synchronize two nodes by extrapolating backwards to estimate the phase shift at a previous time. We provide an implementation on motes that integrates TPSN with post-facto synchronization and gauge its performance on a multihop network of motes. On the other hand, to facilitate deployment of MAC protocols such as TDMA, there might be a need of maintaining a unique and global timescale throughout the network. In this case, we create a self-configuring system, suitable for sensor networks, where a hierarchical structure is established in the network. In our algorithm, NTP-like daemons self-organize to act as servers to some nodes while acting as client to another server. In this paper, we specially consider such scenarios and provide an integrated algorithm that first establishes the hierarchical structure and then aims at providing a unique global timescale throughout the network.

We will show that TPSN provides a simple, scalable and efficient solution to the problem of timing synchronization in sensor networks. Moreover, TPSN is completely flexible and can be easily tuned to meet the desired levels of accuracy as well as algorithmic overhead. TPSN also comes with an auxiliary benefit of improving the accuracy of other basic services in sensor networks such as localization, target tracking, aggregation.

2. RELATED WORK

Although time synchronization has been the topic of research for the past many years in a wide spectrum of applications [5], [6], [9], with regard to sensor networks, a final solution is yet to be found. In this section we will discuss some of the algorithms that deal with timing issues in sensor networks.

In [8], [10], researchers emphasize on a completely different regime for time synchronization in sensor networks. They have pointed out notable differences between timing synchronization requirements in sensor networks as compared to traditional networks. In general, the problem of time synchronization can be studied in context of three different models.

The first and perhaps the simplest type of model concentrate on just maintaining the relative notion of time between nodes. Thus the aim here is not synchronize the sensor node clocks but to generate a right chronology of events. A scheme for sensor networks based on this model was proposed in [11]. The algorithm proposed in [11] is only initiated when events take place in the network. Therefore, such a scheme is not extendible to scenarios where a notion of sensor node clock is required. For example, a scenario where the actual time of occurrence of an event is important or where sensor node clock is used to successfully run MAC protocols.

A more complex model is of maintaining relative clocks. In this model, though every node maintains an individual clock, these clocks are not synchronized with respect to each other. Instead, every node stores information about the relative drift between its clock and the clock of any other node in the network (or with only those nodes with which it desires to maintain a relative clock). A scheme based on this model is Reference-Broadcast Synchronization (RBS) [7]. In RBS, sensor nodes periodically send beacon messages to their neighbors using the network's physical layer broadcast. Recipients use the message's arrival time as point of reference for comparing their clocks. The offset between any pair of nodes, receiving the beacon, is calculated by exchanging the local timestamps. This scheme successfully removes all possible sources of error except the variability in processing delay at the receiver.

Perhaps the most complex model ("*always-on*" model) is the one where every node maintains a clock that is synchronized with respect to a reference node in the network. The aim here is to maintain a global and a unique timescale throughout the network. Although this model consumes maximum energy, it is a superset of all the models. Therefore, if the clocks are absolutely synchronized, they are relatively synchronized too and a right chronology of events can also be detected. In this paper, we provide an integrated algorithm where TPSN aims to provide synchronization following this model. To the best of our knowledge, this is the first work that tries to provide timing synchronization in sensor networks on the basis of "always-on" model. In this model, TPSN will be provided as an API, running continuously on the backend of a sensor node transparent to the application writer.

However, we would like to emphasize that TPSN is not an algorithm restricted to the *"always-on"* model. To show the flexibility of TPSN, we provide an example implementation on motes that integrates TPSN with post-facto synchronization to synchronize node clocks, lying on a multihop network, relatively to each other. RBS also uses post-facto synchronization to provide multihop clock synchronization. In this scenario, the difference between RBS and TPSN is the way in which they carry out the handshake to synchronize a pair of motes. TPSN uses the classical approach of sender-receiver synchronization whereas RBS uses the approach of receiver-receiver synchronization. As mentioned earlier, we shall prove that for sensor networks doing sender-receiver synchronization gives better results than receiver-receiver synchronization.

We take the motivation from NTP that has been largely successful in Internet. However, NTP is computationally intensive. It is completely infeasible to implement NTP on the energy constrained sensor nodes. Moreover, NTP is a fixed algorithm that can operate only in the "always-on" model. Contrary to this, TPSN is flexible to the model used for timing synchronization and can also be easily tuned to meet the desired operating point in energy versus accuracy subspace. NTP has been successfully able to synchronize the clocks in Internet to an accuracy of the order of milliseconds but the time synchronization requirements for sensor networks can be much more stringent than that, ranging into the order of a few microseconds. Our scheme can be viewed as a practical, more accurate and a flexible extension of NTP to sensor networks.

3. SYSTEM MODEL

We have *N* sensor nodes scattered in an area *A*. Every node maintains a 16-bit register as a clock that is triggered by a crystal oscillator. This is the only notion of time that a node has. In this paper, we provide an integrated algorithm that aims at providing time synchronization following the always-on model. Thus, the goal is to establish a common timescale for every node in the sensor network and therefore, synchronize the 16-bit clock for every sensor node. We begin by describing the basic concept of TPSN and proceed to outline the assumptions about the system.

3.1 Basic Concept

The first step of the algorithm is to create a hierarchical topology in the network. Every node is assigned a level in this hierarchical structure. We ensure that a node belonging to level *i* can communicate with at least one node belonging to level *i-1*. Only one node is assigned to level *0*, which we call the **"root node"**. We call this stage of our algorithm as the **"level discovery phase"**. Once the hierarchical structure has been established, the root node initiates the second stage of the algorithm, which is called the **"synchronization phase"**. In this phase, a node belonging to level *i* synchronize to a node belonging to level *i-1*. Eventually every node is synchronized to the root node and we achieve network-wide time synchronization.

In general, a user node that acts as the gateway between the sensor network and the external world can act as the root node. The user node can be equipped with a GPS receiver, in which case the sensor nodes would be synchronized to the physical world. In more hostile environments, where it is impossible to have an external entity, sensor nodes can periodically take over the functionality of being the root node, using some leader election algorithm [12]. Also, neither TPSN nor the "always-on" model restricts the possibility of having multiple root nodes in the network. In this case, islands of time-synchronized nodes will be formed in the network. Further, a scheme such as RBS [7] could be used to maintain a relative clock between the adjacent nodes that lie on the boundary, providing synchronization in the whole network. In this paper, we consider the network to have just one root node.

3.2 Assumptions

We assume that the sensor nodes have unique identifiers. A link level protocol ensures that each node is aware of the set of nodes with which it can directly communicate, also termed as the "neighbor set" of the node. Though there can be uni-directional links in the network, TPSN uses only bi-directional links to do pair wise synchronization between a set of nodes. We also assume that it is possible to create a spanning tree in the network using just these bi-directional links.

In this paper, we have attributed the creation and maintenance of the hierarchical structure as the responsibilities of TPSN. However, many of the sensor network applications rely on in-network processing and require a similar structure for their functionality for example the aggregation tree required for TinyDB [13]. Therefore, creating and maintaining a hierarchical structure should not be considered as an overhead exclusive to TPSN.

4. TIMING-SYNC PROTOCOL FOR SENSOR NETWORKS (TPSN)

4.1 Level Discovery Phase

This phase of the algorithm occurs at the onset, when the network is deployed. The root node is assigned a level *0* and it initiates this phase by broadcasting a *level_discovery* packet. The *level_discovery* packet contains the identity and the level of the sender. The immediate neighbors of the root node receive this packet and assign themselves a level, one greater than the level they have received i.e., level *1*. After establishing their own level, they broadcast a new *level_discovery* packet containing their own level. This process is continued and eventually every node in the network is assigned a level. On being assigned a level, a node neglects any such future packets. This makes sure that no flooding congestion takes place in this phase. Thus a hierarchical structure is created with only one node, root node, at level *0*. A node might not receive any *level_discovery* packets owing to MAC layer collisions. We explain how to handle such special cases in Section 4.3. In this paper, we use a simple flooding mechanism to create the hierarchical structure. Instead, we could have used more accurate minimum spanning tree algorithms. We will show that the choice between the two results in an accuracy versus complexity tradeoff.

4.2 Synchronization Phase

In this phase, pair wise synchronization is performed along the edges of the hierarchical structure established in the earlier phase. We use the classical approach of sender-receiver synchronization [5] for doing this handshake between a pair of nodes. We shall show the merits of this approach to the approach of receiver-receiver synchronization in later sections.

Let us first analyze, how a two-way message exchange between a pair of nodes can synchronize them. Figure 1 shows this message-exchange between nodes '*A*' and '*B*'. Here, *T1, T4* represent the time measured by local clock of '*A*'. Similarly *T2, T3* represent the time measured by local clock of '*B*'. At time *T1*, '*A*' sends a *synchronization_pulse* packet to '*B*'. The *synchronization_pulse packet* contains the level number of '*A*' and the value of *T1*. Node *B* receives this packet at *T2,* where *T2* is equal to $T1 + \Delta + d$. Here Δ and *d* represents the clock drift between the two nodes and propagation delay respectively. At time *T3*, '*B*' sends back an *acknowledgement* packet to '*A*'. The *acknowledgement* packet contains the level number of '*B*' and the values of *T1, T2* and *T3*. Node *A* receives the packet at *T4.* Assuming that the clock drift and the propagation delay do not change in this small span of time, '*A*' can calculate the clock drift and propagation delay as:

$$\Delta = \frac{(T2-T1)-(T4-T3)}{2}; d = \frac{(T2-T1)+(T4-T3)}{2} \ldots(1)$$

Knowing the drift, node *A* can correct its clock accordingly, so that it synchronizes to node *B*. This is a sender-initiated approach, where the sender synchronizes its clock to that of the receiver.

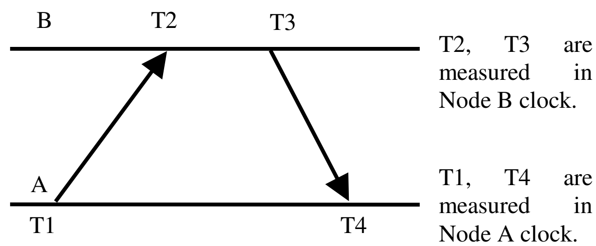

Figure 1: Two-way message exchange between pair of nodes

This message exchange at the network level begins with the root node initiating the phase by broadcasting a *time_sync* packet. On receiving this packet, nodes belonging to level *1* wait for some random time before they initiate the two-way message exchange with the root node. This randomization is to avoid the contention in medium access. On receiving back an acknowledgment, these nodes adjust their clock to the root node. The nodes belonging to level *2* will overhear this message exchange. This is based on the fact that every node in level *2* has at least one node of level *1* in its neighbor set. On hearing this message, nodes in level *2* back off for some random time, after which they initiate the message exchange with nodes in level *1*. This randomization is to ensure that nodes in level *2* start the synchronization phase after nodes in level *1* have been synchronized. Note that a node sends back an *acknowledgement* to a *synchronization_pulse*, provided that it has synchronized itself. This ensures that no multiple levels of synchronization are formed in the network.

This process is carried out throughout the network and eventually every node is synchronized to the root node. In a sensor network packet collisions can take place quite often. To handle such scenario a node waiting for an acknowledgement, timeouts after some random time and retransmits the *synchronization_pulse*. This process is continued until a successful two-way message exchange has been done.

4.3 Special Provisions

In a sensor network, the nodes are usually deployed in a random fashion. So scenarios might exist where a sensor node joins an already established network i.e., the node might join the network when the level discovery phase is already over. Even if the node is present at the onset of the network, it might not receive any *level_discovery* packets owing to MAC layer collisions. In either case it will not be assigned any level in the hierarchy. However, every node needs to be a part of the hierarchical topology so that it can be synchronized with the root node. Thus, when a node is deployed, it waits for some time to be assigned a level. If it is not assigned a level within that period, it timeouts and broadcasts a *level_request* message. The neighbors reply to this request by sending their own level. The new node assigns itself a level, one greater then the smallest level it has received and hence, joins the hierarchy. This could be seen as a *local level discovery* phase.

Sensor nodes may also die randomly. A situation may arise, when a level *i* node does not have any neighbor at level *i-1*. In such scenarios, the node would not get back an *acknowledgement* to its *synchronization_pulse*. Thus, this node at level *i* would not be able to synchronize to the root node. It has already been explained that in order to handle collisions, a node would retransmit the *synchronization_pulse* after some random amount of time. After retransmitting the *synchronization_pulse* a fixed number of times, a node assumes that it has lost all its neighbors on the upper level and broadcasts a *level_ request* message. On getting back a reply, the node is assigned a new level. Assuming the network is still connected, the node will have at least one node in its *neighbor set* and thus it will surely be assigned a new level in the hierarchy. We consider four retransmissions to be a heuristic for deciding non-availability of a neighbor in the upper level. The validity of this heuristic has been verified via simulations. In general, choosing a large number will increase the time taken for synchronization whereas a small number will cause unnecessary flooding in the network, decreasing the synchronization accuracy.

We started with the premise that a node has been designated as the root node. If an elected root node dies, the nodes in level *1* would not receive any *acknowledgement* packets and hence, they will timeout following the scheme described above. Instead of broadcasting a *level_request* packet, they run a leader election algorithm and the elected leader takes over the functionality of the root node. This new root node starts from the beginning and reruns the level discovery phase. Note that these special provisions are essentially heuristics to take care of ambiguities in the network. Though we do not claim that these are indeed the optimal solutions, we have verified their efficacy via extensive simulations.

5. ERROR ANALYSIS OF TPSN

In this section, we characterize the possible sources of error and present a detailed mathematically analysis for TPSN. We concentrate on pair wise synchronization between two nodes. We compare the performance of our scheme to RBS [7], an algorithm that synchronizes a set of receivers in sensor networks. However, as will be clear from our analysis, the results can be in general extended to make a comparison between the classical approach of sender-receiver synchronization and receiver-receiver synchronization.

5.1 Decomposition of Packet Delay

Figure 2 shows the decomposition of packet delay when it traverses over a wireless link between two sensor nodes. We designate the node that initiates the packet exchange as the *sender* and the node that responds to this message as the *receiver*. Although a similar decomposition has also been presented in [7], we detail the various delay components from a systems perspective. In this discussion, we will borrow terms from a typical layered architecture used in traditional computer networks. We briefly analyze each component shown in Figure 2.

- **Send time**: When a node decides to transmit a packet, it is scheduled as a *task* in a typical sensor node. There is time spent in actually constructing the packet at the *application layer*, after which it is passed to the lower layers for transmission. This time includes the delay incurred by the packet to reach the *MAC layer* from the application layer. This delay is highly variable due to the software delays introduced by the underlying operating system.
- **Access time**: After reaching the MAC layer, the packet waits until it can access the channel. This delay is specific to wireless networks resulting from the property of common medium for packet transmission.

This is perhaps the most critical factor contributing to packet delay. Moreover, it's highly variable in nature and is specific to the MAC protocol employed by the sensor node.

- **Transmission time**: This refers to the time when a packet is transmitted bit by bit at the physical layer over the wireless link. This delay is mainly deterministic in nature and can be estimated using the packet size and the radio speed. The software implementation of the transmitter will have a few minor variations due to the response time for interrupts. In [14], a novel hardware based RF transceiver has been proposed where the variations would be completely negligible.
- **Propagation time**: This is the actual time taken by the packet to traverse the wireless link from the sender to the receiver. The absolute value of this delay is negligible as compared to other sources of packet latency.
- **Reception time**: This refers to the time taken in receiving the bits and passing them to the MAC layer. This is going to be mainly deterministic in nature. The variations in reception delay would even be smaller if the sensor node employs a hardware based RF transceiver [14].
- **Receive time**: The bits are then constructed into a packet and then this packet is passed on to the application layer where it's decoded. The time taken in this whole activity refers to receive time. The value of receive time changes due to the variable delays introduced by the operating system.

In Figure2, the size used for different boxes is just to give an intuition about the absolute value of each component. It doesn't correspond to their actual ratio. For example, we expect that the access time (MAC delay) would completely overshadow other delays in practice. Secondly, communication takes place in bits and a node optimizes by performing events in parallel. Thus, when a bit is being coded for transmission, another bit could be in air or being received at the other end simultaneously. Thus, this decomposition is just an approximation when done at the packet level instead of the bit level.

5.2 Error Analysis

In this section, we will contrast TPSN to RBS by analyzing the sources of error for both the schemes. In [7], the efficiency of RBS over NTP has already been shown and therefore one can use the two comparisons to gauge the performance of TPSN to NTP. For this analysis, we introduce the notion of real time i.e. the time measured by an ideal clock as shown in Figure 3. We represent the times measured by local node clocks in Figure 1, such as *T1,* in real time by using lowercase letters. Thus t1 stands for the real time (measured by ideal clock) equivalent of T1 (measured by node A clock). We consider the same scenario as shown in Figure 1. Node A sends a packet at T1 and node B receives it at T2. Note that T1 and T2 are times measured by node clocks of A and B respectively. The following set of equations can be easily derived:

$$t2 = t1 + S_A + P_{A->B} + R_B \ldots (2)$$

$$T2 = T1 + S_A + P_{A->B} + R_B + D_{t1}^{A->B} \ldots (3)$$

Here S_A, $P_{A->B}$, R_B refer to the time taken to send packet (*send time* + *access time* + *transmission time*) at node A, *propagation time* between node A, *B* and time taken to receive packet (*reception time* + *receive time*) at node *B* respectively. All these times are with respect to an ideal clock. Here, $D_t^{A->B}$ refers to the drift between the nodes A and *B* at time *t*. Node B then sends a reply at T3, which is received by node A at T4. Using similar analysis following equation can be derived:

$$T4 = T3 + S_B + P_{B->A} + R_A - D_{t4}^{A->B} \ldots (4)$$

Note $D_{t3}^{B->A} \approx D_{t4}^{B->A} = -D_{t4}^{A->B}$. Further, $D_{t1}^{A->B}$ can be broken into two components as follows:

$$D_{t1}^{A->B} = D_{t4}^{A->B} + RD_{t1->t4}^{A->B} \ldots (5)$$

Here $RD_{t1->t4}^{A->B}$ refers to the relative drift between the nodes A and B from time *t1* to *t4*. Figure 3 pictorially presents the definition of drift and relative drift between the node clocks. In equation 5, $RD_{t1->t4}^{A->B}$ can be positive or negative depending on which node clock leads the other. Subtracting equation 4 from 3 and using equations 1 and 5, we obtain:

$$(2 * \Delta) = S^{UC} + P^{UC} + R^{UC} + RD_{t1->t4}^{A->B} + (2 * D_{t4}^{A->B}) \ldots (6)$$

Here S^{UC}, R^{UC} and P^{UC} stand for the uncertainty at sender, at receiver and in propagation time respectively. They are given by the following equations:

$$S^{UC} = S_A - S_B \ldots (7)$$

$$R^{UC} = R_B - R_A \ldots (8)$$

$$P^{UC} = P_{A->B} - P_{B->A} \ldots (9)$$

The aim is to calculate $D_{t4}^{A->B}$, as we correct the clock at T4 (equal to real time t4) at node A. Rearranging the terms of equation 6, we obtain

$$Error = \Delta - D_{t4}^{A->B} = \frac{S^{UC}}{2} + \frac{P^{UC}}{2} + \frac{R^{UC}}{2} + \frac{RD_{t1->t4}^{A->B}}{2} \ldots (10)$$

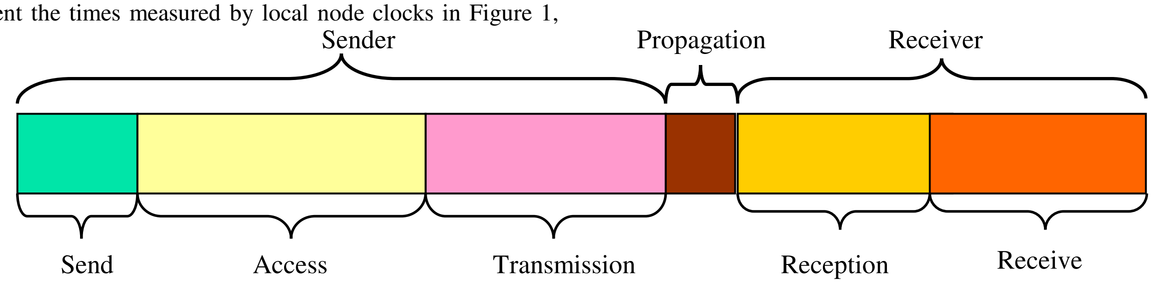

Figure 2: Decomposition of packet delay over a wireless link

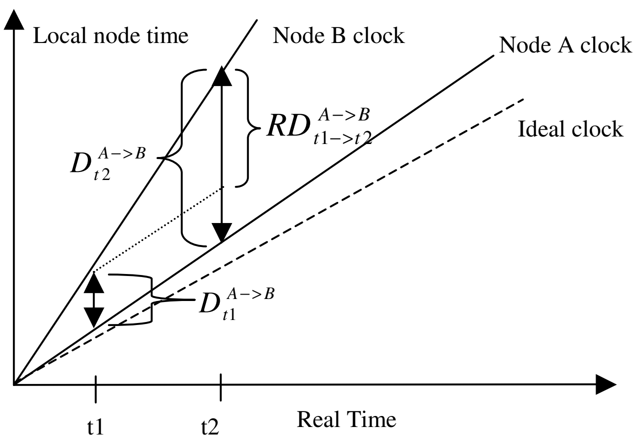

Figure 3: Drift among the local node clocks

TPSN is a sender-receiver synchronization algorithm, whereas RBS is a receiver-receiver synchronization algorithm. In RBS, two receiver nodes exchange timing information about the message that they have received from a common sender. Suppose the two nodes *A* and *B* receive the common packet at *T2, T3* respectively generated by *C* at time *T1*. Thus, by similar break up of packet latency we can obtain the following equations:

$$T2 = T1 + S_C + P_{C->A} + R_A + D^{C->A}_{t1} \ldots (11)$$

$$T3 = T1 + S_C + P_{C->B} + R_B + D^{C->B}_{t1} \ldots (12)$$

Node B sends this timestamp information (T3) in a separate packet, which is received by A at time T4. As mentioned earlier, the aim is to calculate $D^{A->B}_{t4}$. RBS calculates it by subtracting equations 11 from 12. Finally, the expression for error can be developed as:

$$\Delta = (P_{C->B} - P_{C->A}) + (R_B - R_A) + (D^{C->B}_{t1} - D^{C->A}_{t1}) \ldots (13)$$

$$D^{C->B}_{t1} - D^{C->A}_{t1} = D^{A->B}_{t1} = D^{A->B}_{t4} + RD^{A->B}_{t1->t4} \ldots (14)$$

$$Error = \Delta - D^{A->B}_{t4} = P^{UC}_{D} + R^{UC} + RD^{A->B}_{t1->t4} \ldots (15)$$

Here P^{UC}_{D} represents the uncertainty in propagation time between two distinct node pairs and is given by:

$$P^{UC}_{D} = P_{C->B} - P_{C->A} \ldots (16)$$

As can be seen from equations 10 and 15, the two contributing factors towards the synchronization error for both TPSN and RBS are the variation in packet delays and the drift among the local clock of motes. Let us analyze each factor individually.

5.2.1 Variation in packet delays

- **Uncertainty at the sender (S^{UC})**: As can be observed from equation 15, RBS completely eliminates the uncertainty at the sender side. In fact, this is the main reason why many researchers believe that receiver-receiver synchronization performs better than classical sender-receiver synchronization. This is potentially of advantage when the radio and its driver is a closed black box such as in the case of wireless LANS. However in the case of sensor networks there exist a strong coupling between the radio and the application layer. In fact sensor nodes such as motes [15] do most of the radio processing at the application layer. This provides a huge amount of flexibility in sensor networks. We use this to drive our motivation for doing sender-receiver synchronization. We propose to reduce this source of error by time stamping the packet at the MAC layer (i.e. when the packet is about to be transmitted) instead of time stamping the packet at the application layer. Thus, S_A in equation 7 becomes equal to *transmission time*, instead of the total time taken to send the packet at the sender node (*send + access + transmission time*). This means the only erroneous factor that remains is the uncertainty in *transmission time*. As we have mentioned before, we claim that this delay is mainly deterministic in nature. Therefore we expect the resulting contribution of this factor to the net error to be small.
- **Uncertainty in propagation time (P^{UC})**: As can be observed from equation 10 and 15, both TPSN and RBS suffer from the variation in propagation time. TPSN uses only symmetric links and on such links, the variation in delay is going to be negligible. As can be observed from equation 15, in case of RBS, the error is between two distinct node pairs and hence can be large depending on the distances between them. Let us assume the best-case scenario for RBS, when the variation in propagation time is the same as in TPSN, equal to η time units. Using equations 10 and 15, this will result in an error of η/2 and η time units for TPSN and RBS respectively. Thus even for the same variation in propagation time, TPSN is better off by a factor of 2 than RBS.
- **Uncertainty at the receiver (R^{UC})**: By time stamping the packet at the MAC layer, TPSN removes the *receive time* completely. Thus, R_A in equation 8 becomes equal to only the *reception time*, instead of the total time taken in receiving the packet at the receiver (*reception + receive time*). In its proposed form, RBS timestamps the packet at the application layer and thus suffers from variation in both *reception* as well as *receive time*. However, in order to make a fair comparison let us assume that RBS is also implemented with capability of time stamping the packets at the MAC layer. With such a system even in RBS the only source of error will be the variation in *reception time*. However for the same variation in *reception time* of α time units, RBS gives synchronization error of α time units whereas TPSN gives a synchronization error of α/2 time units. This can be observed from equations 10 and 15. Thus, even for a similar system, TPSN provides a 2x better performance as compared to RBS.

5.2.2 Drift among the local clocks ($RD^{A->B}_{t1->t4}$)

The clocks in sensor nodes are envisioned to be crystal based, mainly because of their low cost. Such crystals are susceptible to huge drifts from the ideal clock, as shown in Figure 3. However, as can be observed from the last terms on RHS of equation 10 and 15, all we care about is just the relative drift between the two nodes.

Besides depending on the rate of relative drift between the two nodes, the error performance also depends on the time taken for the completion of the algorithms *i.e.,* the difference of *t1* and *t4*. To make an approximate comparison, let us assume that the two algorithms observe the same variation in drift of β time units. This is quite reasonable as both TPSN and RBS involves two packet transfers between nodes. Using equation 10 and 15, this shall result in a synchronization error of β time units for RBS whereas an error of $\beta/2$ time units for TPSN. Therefore again with the same variation in drift, TPSN provides a 2x better performance as compared to RBS.

5.2.3 Conclusion

As can be seen from the above analysis, TPSN would give roughly a 2x better performance for all the sources of error as compared to RBS. However, TPSN has an added contribution from the uncertainty at the sender whereas RBS completely removes this as a source of error. This analysis can be extended to comparison between sender-receiver and receiver-receiver synchronization based algorithms in general. In case of traditional wireless networks the uncertainty in MAC delay is so large that it completely overshadows the effect of other factors, resulting in giving an edge to algorithms based on receiver-receiver synchronization. However in case of sensor networks, by having the flexibility of time stamping the packets at the MAC layer, we remove this critical source of error. As a result, we believe that the classical approach of doing sender-receiver synchronization is a better approach of doing time synchronization than receiver-receiver synchronization in sensor networks.

We have shown this via a detailed analysis and we verify our claim by implementing TPSN and RBS on motes.

6. IMPLEMENTATION ON BERKELEY MOTES

In this section we describe a prototype system that we built around Berkeley Motes [15] to implement TPSN. In [7], authors verify the efficacy of RBS by implementing it on an IPAQ-motenic testbed. The mote is used as a nic (network interface card) to the IPAQ. Rest of the protocol stack, the time synchronization algorithm as well as the clock maintenance runs on an IPAQ. We present here a more practical implementation of RBS on stand-alone sensor nodes (Motes) without using any external components.

In [7], authors have reported numbers on synchronization error that has an absolute magnitude much less than in this paper (6.5µs). This is just an artifact of using a superior operating system (Linux) and much more stable crystals available in IPAQs. The only system requirement that TPSN wants is the capability of time stamping the packet at the MAC layer. We don't see any other system effects that will degrade/improve the performance of RBS more than that of TPSN. Therefore, the relative performance of TPSN with RBS would continue to be the same, if instead an IPAQ-motenic test-bed is used.

6.1 Overview of Berkeley Motes

MICA motes [15] are second-generation wireless modules used for research of low power wireless sensor networks. The devices have application in research, new security applications, environmental monitoring, and large-scale distributed networks. MICA Motes run UC Berkeley's open source Tiny OS Operating System [16]. In general sensor node architecture consists of five major modules: processing, RF communication, power management, I/O expansion, and secondary storage. In mica motes, the main controller is an ATMEGA103L running at 4Mhz. There is an AT90LS2343 included to handle wireless reprogramming. An Atmel AT45DB041B serial flash chip provides persistent data storage. The RF module consists of an RF Monolithics RFM TR100 transceiver and can operate at communication rates up to 115Kbps. The system is designed to operate off an inexpensive battery that produces between 3.2V and 2.0V (e.g., pair of AA batteries). More details on the system architecture of motes can be found in [15]. Although several advancements in communication stacks have been proposed, we are using the Berkeley communication stack proposed in the original paper [15].

6.2 Modifying TinyOS

The first task was to generate a lower granularity clock in motes. As mentioned earlier, a crystal oscillator triggers the clock in motes. The maximum frequency of the crystal used in motes is 4Mhz implying that we can achieve a minimum granularity of 0.25µs. In its current form, Tiny OS uses timer 0 (8 bit timer) for providing clock [16]. This made it impossible to generate such a low granularity clock, as the register overflows very quickly generating frequent interrupts. There is only one 16-bit timer in Atmel 103 mode, which is used for sampling the radio. After few modifications, we were able to utilize the same timer in parallel to generate the clock. We maintain a 16-bit register as the clock, which gets triggered by the overflow of timer 1. We run timer 1 at the maximum frequency of 4Mhz.

The second major modification needed was to incorporate the ability of time stamping the packets at the RFM layer (MAC layer). We were able to achieve this by creating an interface between the application layer and the RFM layer. We exclude the details here for space constraints but the readers can find all the details from the source codes [17].

6.3 Synchronizing a Pair of Motes

We first wanted to test the contribution of *uncertainty at the sender* towards the synchronization error. Recall that this was the only factor, which provided degradation in the performance of TPSN as compared to RBS. Our claim was that this factor contributes negligibly to the net error, as a result of which TPSN shall give roughly a 2x better performance than RBS.

The set up consisted of two motes. Each mote maintains a 16-bit clock based on its crystal oscillator. The two motes were started randomly, so that they have completely unsynchronized clocks at the beginning. One of the motes was designated as the sender and was responsible for initiating the message exchange after it has measured 10s in its clock. We measure the *transmission time* (as defined in the earlier section) at both the motes. Figure 4 plots the uncertainty at the sender (S^{UC}), calculated as the difference of transmission times between the two motes, for 100 different simulation runs. We have shown here only the magnitude of the S^{UC}. Using equation 10, it can be observed that a difference in transmission time (S^{UC}) of δ time units contributes a synchronization error of $\delta/2$ time units.

The average magnitude of S^{UC} was around 1.15µs. This implies that on an average, the uncertainty at sender contributes a synchronization error of around 0.62µs. As will be seen from results in the next section, this number is very small as compared to the absolute value of the total synchronization error. This verifies our claim and provides a strong edge to sender-receiver synchronization as compared to receiver-receiver synchronization in general.

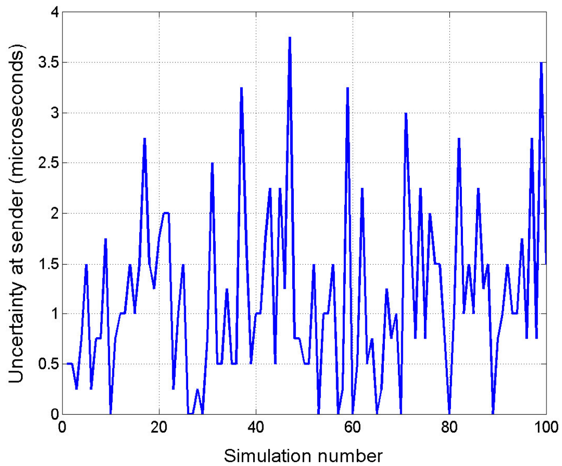

Figure 4: Uncertainty at sender (only magnitude)

The next step was to calculate the absolute synchronization error between the two motes. A similar set up was used so that the two motes (A and B) start randomly with completely unsynchronized clocks. We programmed the two motes so that they continuously toggle a particular output pin after every 8ms. After completing the message exchange, the synchronization error is calculated by observing the phase shift between the two waveforms (corresponding to A and B) on a Digital Analyzer. We have also implemented RBS under the same set up (i.e. time stamping is done at the MAC layer). In case of RBS, a third mote is used and is designated to act as the common sender. We calculate the synchronization error between the same pair of motes. After overhearing the message from the common sender, one of the receivers (node B) transmits this information to the other receiver (node A). Node A corrects its clock by calculating the drift as mentioned in equation 15. We again calculate the synchronization error by observing the phase shift between the two waveforms (corresponding to A and B).

Figure 5 plots the histogram of the synchronization error and the results are summarized in table 1. The results are obtained after averaging over 200 independent runs for both TPSN and RBS. Note that to calculate the statistics we use only the magnitude of the synchronization error and neglect the sign (which clock is ahead among the two nodes). As can be observed from Table 1, we were able to synchronize a pair of motes to an average accuracy of less than 20µs. The best case was the two motes getting perfectly synchronized and the worst-case synchronization error was around 50 µs. Approximately 65% of the times, the error was either equal or smaller than the average synchronization error. As expected, under similar scenario TPSN roughly gives a 2x better performance than RBS. In [7], it was shown that the synchronization error between two motes could be modeled as a normal distribution. This is consistent with the shape of the histogram in Figure 5. The mean of the distribution for RBS will be twice the mean of the distribution for TPSN. In general, synchronization error for TPSN and RBS will be sample points on their respective distributions and hence, it is not necessary that for every simulation run TPSN will give exactly a 2x better performance than RBS.

Table 1: Statistics of Synchronization error (only magnitude)

	TPSN	RBS
Average error (in µs)	16.9	29.13
Worst case error (in µs)	44	93
Best case error (in µs)	0	0
Percentage of time error is less than or equal to average error	64	53

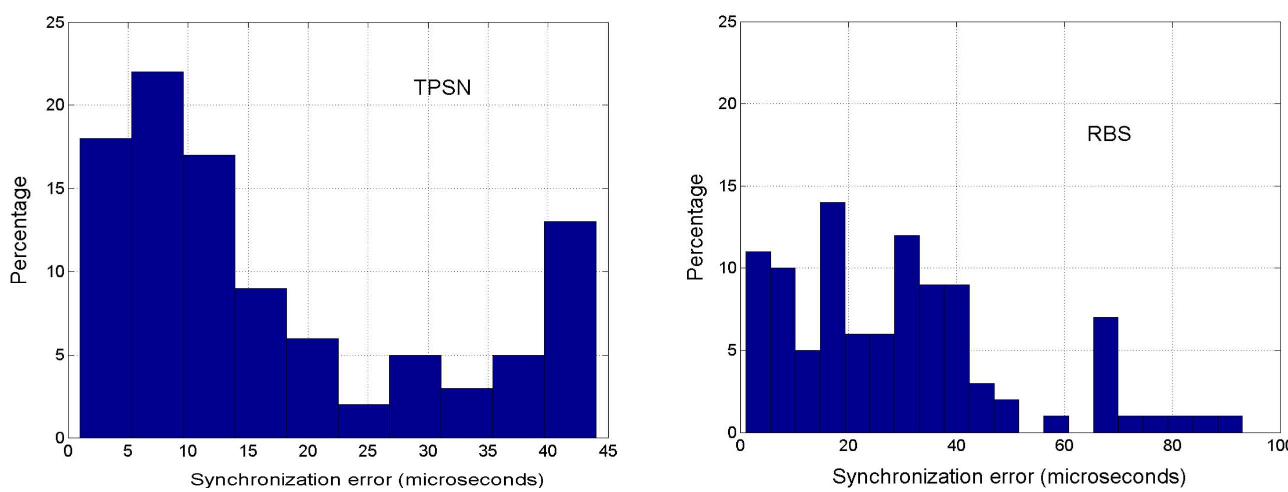

Figure 5: Histogram of Synchronization error (only magnitude)

6.4 Multihop Results

Till now we have presented TPSN in form of an integrated algorithm that aims at providing synchronization following the “always-on” model. However, their exist sensor network applications where time synchronization will be needed over a subset of nodes and that too for a small period of time. To model this scenario, the approach of post-facto synchronization was proposed in [8]. Post-facto synchronization is used to synchronize two nodes by extrapolating backwards to estimate the phase shift at a previous time. Thus the nodes synchronize only when they need to *i.e.,* after an event has been detected. Consider an example scenario where the objective is to send a packet from source A to sink F via path A->B->C->D->E->F and to simultaneously synchronize them. Thus at every hop, after receiving the packet, the receiver synchronizes its clock using TPSN to the sender, before sending it to the next hop. For example at first hop, B synchronizes its clock to A and after that forwards the packet to C. Thus eventually every node in the path would have the its clock synchronized with respect to node A. At every hop we deliberately introduce a delay of 2 seconds to model the packet processing time as well as the large MAC delay that a node can potentially suffer in a large-scale network. Every mote runs CSMA and the 2 seconds doesnot include the random back off time for detecting the channel to be idle. We implement this integration of TPSN with post-facto synchronization approach on Berkeley motes.

Figure 6 plots the average synchronization error versus the hop distance. We also plot the histogram, showing the synchronization error observed for every individual pair. The results were averaged over every possible pair. For example in order to calculate the error for 2 hop distance, the average was taken over 4 possible mote pairs {(A, C), (B, D), (C, E), (D, F)}. For every possible pair, we have taken 100 independent measurements and while calculating the average, we only consider the magnitude of the error. The synchronization error was measured in a similar way, by measuring the phase shift on the Digital Analyzer between the pair of motes being considered. We only consider those runs where the packet was finally delivered to the sink. As can be seen from Figure 6, the error does not blows up with hop distance. In fact for this scenario it seems that error almost becomes a constant beyond 3-hop distance. This is because synchronization error between any pair of motes will probabilistically take different values from the normal distribution obtained in the earlier section. The randomness in the sign as well as the magnitude of the synchronization error and drift prevents the error from blowing up. If all the motes have been drifting in the same direction and the error was a deterministic quantity, the error would have blown up with the number of hops. Note that we are not claiming that the error cannot blow up. In the worst case, it can increase linearly with the hop distance. However the probability of that event is very low. In fact, we have observed instances where the error value was large. To present a clear picture we detail the obtained statistics in Table 2. We are currently in the process of developing an analytical model for multihop error.

In a similar scenario, RBS would have also used the approach of post-facto synchronization to synchronize the nodes along the multihop path. However, unlike TPSN, RBS will do the handshake between a pair of nodes using receiver-receiver synchronization. As shown earlier, the two neighboring nodes will be synchronized to an error that is on an average 2 times more than TPSN. The errors over multihop can be assumed to be independent and thus, we can conclude that the worst-case mean error for an n-hop network will be 2*n times more in RBS as compared to TPSN. Similarly, the best case mean error for an n-hop network will be 2 times more in RBS as compared to TPSN. Though this gives a range to look for the expected performance, the probability for both worst and best case will be negligibly small. It is hard to conclude anything about the average error, as we do not have a model to characterize the multihop error for either TPSN or RBS. In general we expect that the performance of TPSN will be better than RBS by a factor that lies in the range of 2-2n for an n-hop network. Note that we have made these speculations using the independence model for synchronization error over multihop. The basic design of RBS makes it easy to exploit the concept of multi path diversity among nodes to improve the accuracy performance. In its proposed form, TPSN operates on a fixed infrastructure (hierarchical structure) and is unable to exploit this diversity. We are currently working towards a more sophisticated model to exploit this multipath diversity for TPSN.

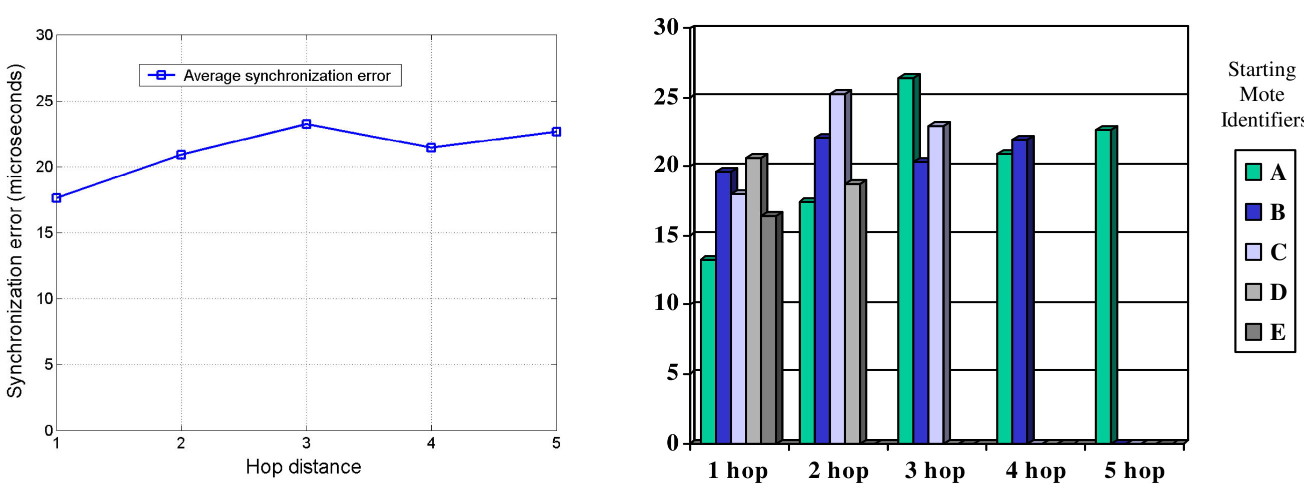

Figure 6: Synchronization error over multihop (only magnitude)

Table 2: Statistics of synchronization error over multihop (only magnitude)

	1 hop distance	2 hop distance	3 hop distance	4 hop distance	5 hop distance
Average error (in µs)	17.61	20.91	23.23	21.436	22.66
Worst case error (in µs)	45.2	51.6	66.8	64	73.6
Best case error (in µs)	0	0	2.8	0	0
Percentage of time error is less than or equal to average error	62	57	63	54	64

6.5 Need for Resynchronization

Though the skew and the drift among the crystal based clocks of sensor nodes can be bounded in general, the range of their deviation can be large. For Berkeley motes, the upper bound given in the datasheets [16] is 40ppm i.e. a clock in mote can loose up to 40µs in a second. A more detailed description of the functionality of mote's clock can be found in [15]. Hence, even if we synchronize the whole network once, nodes will go out of sync in a few minutes. Thus to establish acceptable levels of accuracy in sensor networks at every instant of time, there is a need of doing periodic synchronization. Although sensor node clocks are susceptible to huge drift with respect to the ideal clock, the synchronization error is just a function of relative drift between the nodes and not of the actual drift with the real time clock.

To put this into perspective, we calculate the value of the relative drift between mote (A) and 5 other motes (B, C, D, E and F respectively). In order to calculate the drift between a pair of motes, we run TPSN and measure the synchronization error between the motes at periodic intervals of 1 minute. Figure 7 plots the obtained results that have been averaged over 10 independent runs for every pair of motes. Unlike previous cases, while taking the average we do take into account the sign of the synchronization error. The time *0* on x-axis represents the time when the motes get synchronized for the first time. Note that value of synchronization error at time *0* is not *0*. It's just an artifact of the scale chosen for the plot. As can be observed from Figure 7, the relative drift between every pair of mote increases linearly with time. However it is difficult to conclude any consistent mathematical model for the drift among the motes. In this case, the worst-case relative drift between any pair comes out to be around 4.75µs/s. We have carried out a number of experiments but we have still not came across a pair of motes that have relative drift worse than this number.

The period of TPSN can be calculated with the knowledge of this relative drift and the desired accuracy bound. Consider a hypothetical example where the desired worst-case accuracy bound between a pair of neighboring motes in the network is 10ms. As can be observed from Table 1, using TPSN the worst-case synchronization error between a pair of motes is around 50µs. If the worst-case drift between the motes is 4.75µs/s, the period of TPSN (x) can be calculated as:

$$10*10^{-3} = 50*10^{-6} + (4.75*10^{-6}*x); x \approx 34 \min .$$

6.6 Asymptotic Analysis of Synchronization Error

When TPSN is run periodically, much tighter bounds on accuracy can be obtained by keeping into account the past history. Suppose at the *nth* cycle the node averages the drift over the past *n* cycles, than the error will be given by:

$$Error = \{\frac{1}{n}\sum_{i=1}^{n}\Delta_i\} - (D_{t4}^{A->B})_n \ldots (16)$$

The subscript represents the cycle number. Since it is a periodic activity, we can assume that the two nodes would approximately drift apart by the same value between the two cycles. Thus, at every cycle the value to be estimated, $D_{t4}^{A->B}$, remains the same, except perhaps the first cycle. Thus equation 16 can be rewritten as:

$$Error = \frac{1}{n}\sum_{i=1}^{n}\{\Delta_i - (D_{t4}^{A->B})_i\} \ldots (17)$$

$$Error = \frac{1}{n}\sum_{i=1}^{n}\{\frac{S_i^{UC}}{2} + \frac{P_i^{UC}}{2} + \frac{R_i^{UC}}{2} + \frac{(RD_{t1->t4}^{A->B})_i}{2}\} \ldots (18)$$

Using law of large numbers, as n tends to infinity:

$$\lim_{n->\infty} Error = \frac{E[S^{UC}]}{2} + \frac{E[P^{UC}]}{2} + \frac{E[R^{UC}]}{2} + \frac{E[RD_{t1->t4}^{A->B}]}{2} \ldots (19)$$

Here, *E[.]* stands for the expected value. There seems no reason to believe that uncertainty in transmission time; propagation time and reception time would be a non-zero mean process. Therefore we expect the error to converge asymptotically to $E[RD_{t1->t4}^{A->B}]$ / 2. Packet exchanges over motes take time of the order of milliseconds and therefore we expect this value to be really small.

Another way of interpreting Equation 17 is that the synchronization error after the *nth* cycle will be the error averaged over all the past *n* cycles. We claim that this error will be very small. A way of verifying this claim is to calculate the average error (with its sign) over *n* independent runs. However, unlike previous scenarios, instead of randomly starting the two motes, we start the sender mote exactly 2 seconds after starting the receiver mote. This will make sure that the value to be estimated, $D_{t4}^{A->B}$, remains the same, which is a key assumption in this analysis. We do this for five different mote pairs keeping the sender to be mote A. We average out the results for 10 independent runs (*n=10)* for every mote pair. The error values obtained are 5.12µs, -3.96µs, -4.06µs, -0.2µs and –5.47µs respectively. Thus, there is almost a three-five-fold decrease in the value of synchronization error between two neighboring motes. Though the initial results look encouraging, more work needs to be done to concretize our claim. We plan to investigate this is in more detail over a large-scale network of motes.

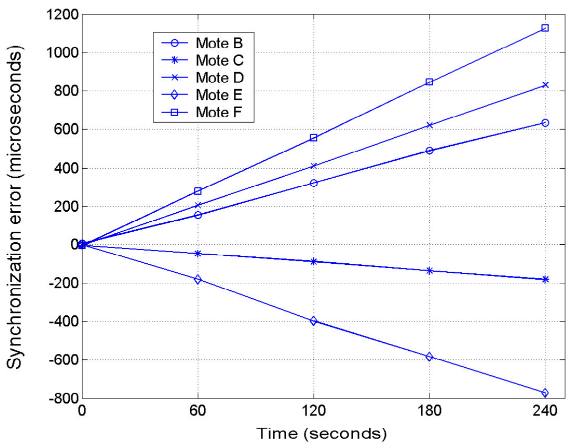

Figure 7: Relative drift with respect to mote A

6.7 Results over Large Scale Networks

We are in the process of creating a test-bed of around 50 motes in our research lab. We envision implementing TPSN over the testbed and verifying its efficacy. However, to gauge the performance of TPSN over large-scale networks we have simulated it over NESLsim [18], a PARSEC based simulation platform for sensor networks. The simulated topologies consist of nodes varying from 150 to 300. Due to space constraints, we just briefly summarize the results in this paper. In [19], the results along with simulation graphs have been documented in form of a technical report.

Using TPSN, each node carries out the handshake with its parent in the hierarchical structure independently of the presence of other nodes in the network. Therefore the synchronization error between two neighboring nodes in the network is independent of the total number of nodes in the network. However, the synchronization error of a node with respect to the root node depends on its hop distance from it. This is because error adds up over multihop. The results obtained in simulations were consistent with this speculation. Although the randomness in the sign and magnitude of the error and the drift prevents it from blowing up, the synchronization error is still a non-decreasing function of the hop distance. Further, the worst-case scenario of linear increase with the hop distance was never observed. A key observation was that the synchronization error of a node depends only on its hop distance and is almost independent of the total number of nodes in the network. Further, the energy consumed by every node to synchronize itself was also measured to be a constant with the increase in number of nodes. This clearly proves that TPSN is scalable with the number of nodes in the network. In its current form, RBS doesnot provide a version for providing network-wide synchronization. However TPSN can be replaced by RBS at the synchronization phase to achieve network-wide time synchronization. Thus, now the handshake along the edges of the spanning tree is performed using receiver-receiver synchronization. Analogous to multihop error analysis, it can be easily observed that even for large scale-networks TPSN would provide a superior performance than RBS.

The above discussion clearly highlights the fact that the network-wide performance of TPSN depends on the efficiency of the hierarchical structure. If we can establish a shortest path to the root node for every node, we shall obtain optimal results. In this paper, we have used simple flooding at the level discovery phase because of its simplicity and lower algorithmic overhead. The total energy taken for establishing this hierarchical structure was found to be a constant when we varied the total number of nodes, making it scalable with the number of nodes. Instead, we could have used a minimum spanning tree algorithm and attained a better accuracy at the cost of complexity and higher algorithmic overhead (*O(N)*).

On the basis of these preliminary results, we claim that TPSN will be able to provide a stable, large-scale time synchronization in a multihop sensor network. Note that all the simulation results were obtained in the absence of any external traffic. We are currently investigating the performance of TPSN in presence of data traffic and also sudden topological failures.

6.8 Auxiliary Benefits of TPSN

Just like time synchronization, a critical problem in sensor networks is to let every node know of its location. TPSN comes with an auxiliary benefit of improving the performance of localization service in sensor networks. A common approach of doing localization in sensor networks is to use ultrasonic ranging [20]. The distance between two nodes is calculated by measuring the time of flight of sound. In [20], authors abstract the timing synchronization part by subtracting the time they receive the radio signal, from the time of receipt of the ultrasonic signal. With this approach, they are able to achieve an average ranging accuracy of around 2cm. If instead, the sender mote first synchronizes with the receiver mote using TPSN and then sends the ultrasonic signal, we would be able to achieve much tighter bounds. As we have already shown, we can synchronize a pair of motes to an average accuracy of less than 20 µs. This can be directly converted into ranging accuracy as follows:

Average ranging error ≅ (Speed of sound) * (Average timing synchronization error)

≅ (345m/s)*(20µs) = 0.69cm

Moreover, even the worst case ranging error-using TPSN (345m/s * 50µs ≅ 1.725cm) is better than the average accuracy obtained in [20]. This clearly magnifies the applicability of TPSN algorithm in solving real time sensor network problems.

7. CONCLUSIONS

Time synchronization is an indispensable piece of infrastructure in sensor networks. In this paper, we have introduced an algorithm, TPSN, for network-wide time synchronization in sensor networks. TPSN is based on the simplistic approach of conventional sender-receiver time synchronization. We argue that unlike traditional wireless networks, for sensor networks classical approach of sender-receiver synchronization is better than receiver-receiver synchronization. We demonstrate our claim by comparing TPSN with an algorithm based on receiver-receiver synchronization, RBS. We show that TPSN roughly gives a 2x better performance than RBS through analysis. We verify the efficacy of this analysis by implementing TPSN and RBS on Berkeley motes. We show that TPSN is completely flexible to the model used for time synchronization. We show an integration of TPSN with post facto synchronization and use this to obtain results on multihop clock synchronization on a network of motes. The efficacy of TPSN for large-scale networks is verified via simulations in PARSEC. The results clearly show that TPSN is completely scalable. Thus, TPSN is a simple, efficient, scalable and a comprehensive solution to the problem of time synchronization in sensor networks.

8. ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation (NSF) under Grant No. ANI-0085773 and by the Office of Naval Research (ONR) under the AINS Program. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the NSF or the ONR.

9. REFERENCES

[1] K. Sohrabi, J. Gao, V. Ailawadhi, G. Pottie, "Protocols for self-organization of a wireless sensor network," *IEEE* Personal *Communications Magazine*, Vol.7, No.5, pp. 16-27, Oct. 2000.

[2] D. Estrin, R. Govindan, J. Heidemann, S. Kumar, "Next Century Challenges: Scalable Coordination in Sensor Networks", *ACM Mobicom Conference*, Seattle, WA, August 1999.

[3] S. I. Roumeliotis, G. A. Bekey, "An extended kalman filter for frequent local and infrequent global sensor data fusion," *In Proc. of the SPIE (Sensor Fusion and Decentralized Control in Autonomous Robotic Systems,* Pittsburgh, PA, USA, Oct. 14-15, 1997, pp.11-22.

[4] V. Claesso, H. Lönn, N. Suri, "Efficient TDMA Synchronization for Distributed Embedded Systems" *20th symposium on Reliable Distributed Systems (SRDS)*, pp 198-201, October, 2001.

[5] D. L. Mills, "Internet time synchronization: The Network Time Protocol" In Z. Yang and T.A. Marsland, editors, *Global States and Time in Distributed Systems.* IEEE Computer Society Press, 1994.

[6] P. Verissimo, L. Rodrigues, A. Casimiro, "CesiumSpray: A Precise and Accurate Global Time Service for Large-Scale Systems," *Journal of Real-Time Systems,* 12(3): 243-294, May 1997.

[7] Jeremy Elson, Lewis Girod and Deborah Estrin, "Fine-Grained Network Time Synchronization using Reference Broadcasts," *In the proceedings of the fifth symposium on Operating System Design and Implementation (OSDI 2002),* December 2002.

[8] J. Elson, D. Estrin, "Time Synchronization for Wireless Sensor Networks," *Proceedings of the 2001 International Parallel and Distributed Processing Symposium (IPDPS), Workshop on Parallel and Distributed Computing Issues in Wireless and Mobile Computing*, San Francisco, California, USA, April 2001.

[9] K. Arvind, "Probabilistic Clock Synchronization in Distributed Systems," *IEEE Transactions on Parallel and Distributed Systems,* 5(5): 474-487, May 1994.

[10] J. Elson, K. Romer, "Wireless Sensor Networks: A New Regime for Time Synchronization," *Proceedings of the First Workshop on Hot Topics In Networks (HotNets-I),* Princeton, New Jersey. October 28-29 2002.

[11] Kay Romer (ETH-Zurich), "Time synchronization in ad hoc networks," *Mobihoc,* 2001.

[12] N. Malpani, J. L. Welch, N. Vaidya, "Leader election algorithm for mobile ad-hoc networks," *In Proceedings of 4th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communication*, pp. 96-103, August 2000.

[13] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "TAG: A Tiny AGgregation Service for Ad-hoc Sensor Networks", *OSDI Conference*, 2002.

[14] Chipcon CC1000 Radio Datasheet, http://www.chipcon.com/files/CC1000_Data_Sheet_2_1.pdf

[15] J. Hill and D. Culler, "A Wireless Embedded Sensor Architecture for System-level Optimization." *Technical report*, U.C. Berkeley, 2001.

[16] TinyOS, http://webs.cs.berkeley.edu/tos/

[17] S. Ganeriwal, Mani B. Srivastava, "Timing-sync Protocol for Sensor Networks (TPSN) on Berkeley Motes", NESL 2003.

[18] S. Ganeriwal, V. Tsiatsis, C. Schurgers, M. B. Srivastava, "NESLsim: A parsec based simulation platform for sensor networks, " NESL, 2002.

[19] S. Ganeriwal, R. Kumar, S. Adlakha, M. B. Srivastava, "Network-wide time synchronization in sensor networks," NESL Technical Report, 2003.

[20] A. Savvides, C. C. Han, M. B. Srivastava, "Dynamic fine-grained localization in ad-hoc networks of sensors", *MobiCom 2001*, Rome, Italy, pp.166-179, July 2001.