TSync : A Lightweight Bidirectional Time Synchronization Service for Wireless Sensor Networks

Hui Dai **Richard Han**
{*huid, rhan*}*@cs.colorado.edu*
Department of Computer Science
University of Colorado at Boulder
Boulder, Colorado, USA

Time synchronization in a wireless sensor network is critical for accurate timestamping of events and fine-tuned coordination of wake/sleep duty cycles to reduce power consumption. This paper proposes TSync, a novel lightweight bidirectional time synchronization service for wireless sensor networks. TSync's bidirectional service offers both a push mechanism for accurate and low overhead global time synchronization as well as a pull mechanism for on-demand synchronization by individual sensor nodes. Multi-channel enhancements improve TSync's performance. We deploy a GPS-enabled framework in live sensor networks to evaluate the accuracy and overhead of TSync in comparison with other in-situ time synchronization algorithms.

I. Introduction

Wireless sensor networks (WSNs) have recently emerged as an important and growing research area. Typically, a WSN consists of a large number of nodes that sense the environment and collaboratively work to process and route the sensor data [11][15]. The architecture of WSNs is typically characterized by hierarchy, where a base station acts as a gateway that collects sensor data and relays the data over a wired backbone to a back-end server for further processing. Application scenarios for such WSNs are wide-ranging, and include battlefield monitoring [3], robotic toys [8], as well as habitat monitoring [1][2].

Time synchronization is an important issue in the correct operation of deployed sensor networks. First, it is often critical to keep a globally synchronized clock when a sensor reading is taken in order to determine the right chronology of events. The lack of a global clock will result in inaccurate timestamping as the local clocks drift on each sensor node. As a base station collects sensor data, inaccurate time stamps from different sensor nodes can lead the base station to falsely reorder, or even reverse, an actual sequence of events. Second, time synchronization has been found to be crucial for efficiently maintaining low duty cycles in sensor networks [2]. The majority of the lifetime of a sensor network should be spent sleeping to conserve energy. During the brief wake periods, neighboring sensor nodes should be synchronized to be awake together so that packet messages can be quickly routed between neighbors and over multiple hops to the base station. If the sleep times are unsynchronized or random, then packets containing sensor event data may be slow to propagate through the sensor network, because neighbors may be asleep and unable to relay messages.

Time synchronization in WSNs is faced with a variety of challenges. First, the resource constraints imposed by WSNs both in terms of limited battery life and limited bandwidth necessitate that any algorithm achieve time synchronization in a lightweight manner, i.e. with low packet transmission overhead so that the radio does not expend excessive energy and bandwidth sending synchronization packets. Second, the broadcast nature of the wireless medium introduces packet collisions between sensors as well as lost packets. This increases the variance in the delay experienced by routed packets. The inaccuracy of time synchronization algorithms developed for wired networks increases with delay variance. As a result, new algorithms are needed to achieve time synchronization over wireless multi-hop sensor networks. Third, the sensor network consists of inexpensive nodes that use low cost crystals to provide the clock. These inexpensive clocks are far more susceptible to clock drift at unknown rates than traditional resource-rich laptops and servers. Finally, different applications will have different clock precision requirements. For some applications, loose synchronization that maintains just the relative timing order may be satisfactory, while other applications may only require a precision of tens of milliseconds. However, in cases such as localization, synchronization accuracy on the order of microseconds is required for location determinination and range-finding. To accommodate this range of application requirements, a time synchronization service will need to be flexible.

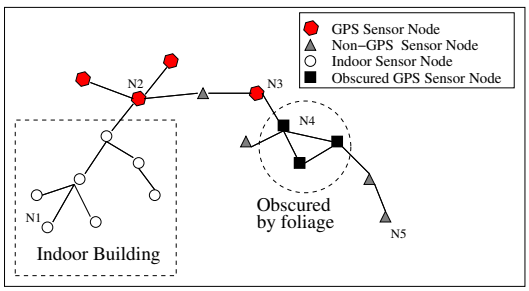

Figure 1: GPS-enabled sensor networks require time synchronization for obscured outdoor or indoor nodes.

Potential time synchronization approaches for sensor

networks include the Global Positioning System (GPS) [20], which is capable of providing a globally accurate clock to each node in a WSN. One example of a GPS-enabled outdoor sensor network is Zebranet for wildlife monitoring [21]. However, GPS has an inherent weakness, namely the requirement of line-of-sight to orbiting satellites. As a result, GPS-enabled time synchronization is inappropriate for indoor WSNs, though assisted GPS via pseudolites is attempting to provide indoor coverage [22]. GPS-enabled time synchronization is also insufficient outdoors even in a sensor network where all nodes possess GPS capability. As shown in Figure 1, some sensor nodes may be obscured, e.g. by foliage, buildings, or mountains. Such obscured nodes will still require a distributed time synchronization algorithm in order to obtain an accurate clock from another node in the WSN. In the figure, obscured indoor and outdoor sensor nodes could retrieve accurate clocks from an unobstructed GPS backbone in the outdoor portion of the network.

Several time synchronization protocols have been proposed for wireless sensor networks [13][25][18]. Elson et al proposed Reference Broadcast Synchronization (RBS) [13][14], where a node periodically broadcasts wireless beacon messages to its neighbors. The neighboring nodes use the broadcast beacon's arrival time as the reference point for comparing the clock readings. The local timestamps are exchanged between neighboring peers in order to calculate the drifts and thereby synchronize clocks. RBS removes several non-deterministic sources from traditional time synchronization, and is able to achieve a precision of 1 μsec after thirty broadcasts. A disadvantage of this approach is the overhead caused by the broadcasts exchanged between neighbors to achieve pair-wise synchronization. This overhead increases with network density [24]. An extension to RBS for time synchronization over multiple hops has also been proposed [14]. For two nodes in different domains to synchronize to one another, a multi-hop chain of time-synchronizing gateways needs to be constructed between them in order to exchange the time stamps.

Ganeriwal et al proposed a hierarchical time synchronization protocol for WSNs [25]. The protocol is divided into two different phases, i.e. the level discovery phase and the synchronization phase. In the level discovery phase, each node in the sensor network is assigned a level. The node that initiates the synchronization is called the root node and is assigned level 0. The level field on each node reflects the hop count from itself to the root node. In the synchronization phase, every node exchanges time stamps with its parent in a manner similar to the Simple Network Time Protocol (SNTP) [29]. The root node, which is the parent of all other nodes, provides an accurate reference, and signals its children to initiate SNTP with itself by sending a time synchronization pulse. Sichitiu et al focus on the development of a deterministic time synchronization algorithm for data fusion [18]. The work is evaluated within an 802.11b multi-hop ad-hoc network. Hill et al. claim 2 μsec precision for time synchronization on MICA sensor motes within a single broadcast domain, though the details are vague [16].

Time synchronization has been extensively studied for the traditional Internet and for distributed systems [6][7][5][17][23][19]. However, complex protocols such as NTP [7] are inappropriate for deployment in sensor networks, due to their intensive computational requirements. Several new approaches explore time synchronization over wireless links, e.g. the CesiumSpray system [23] and [19]. These approaches exploit the broadcast nature of the wireless medium, as TSync does. The goal of the CesiumSpray system is also similar to TSync as it tries to provide global time synchronization to large scale real-time systems. However, CesiumSpray requires all hosts to exist in the same broadcasting domain.

The first contribution of this paper is the development of a time synchronization service, i.e.TSync, that is adapted for wireless sensor networks and satisfies the following properties:

- accurate
- lightweight
- flexible
- comprehensive

Sections II, III, and IV describe how TSync manages to achieve these properties using a bidirectional multi-channel approach. Section VI provides an analysis of TSync's performance, including its accuracy and overhead.

A second contribution of this paper is the development and testing of an in-situ GPS-enabled evaluation framework to assess the accuracy of TSync and other time synchronization algorithms in live deployed WSNs. Section V describes this evaluation framework.

II. Bidirectional Time Synchronization Service

II.A. Design Themes

To achieve its goals of an accurate, lightweight, flexible, and comprehensive time synchronization solution for WSNs, TSync adopts several design themes. These themes also address the challenges of wireless communication and resource constraints characteristic of sensor networks.

To achieve an accurate and lightweight solution, a key design theme of TSync is its reliance on exploiting multi-channel radios for frequency diversity. Multi-channel frequency diversity has long been used in wireless communication to reduce packet collisions/interference and prevent jamming [4][26]. Sensor nodes equipped with radios that are capable of communicating on more than one frequency channel are becoming increasingly common in today's sensor networks, e.g. the MICA2 Mote [27], MIT cricket Indoor Location System [28] and the MANTIS Nymph [9], all use the CC1000 multi-channel radio. Such multi-channel radios help minimize packet collisions by allowing adjacent nodes to transmit on different channels. One consequence of reduced collisions is an improvement in the accuracy of time synchronization, since reduced collisions

decrease the variance in roundtrip delay that directly affects the accuracy of clock estimation. A second important consequence is lightweight overhead, since time synchronization probes need not be retransmitted. TSynch achieves even more lightweight operation via efficient message exchange, as described in later subsections.

To achieve a comprehensive and flexible solution, a second key design decision of TSync was to adopt a bidirectional approach. TSync consists of two mechanisms for synchronization: a push-based mechanism and a pull-based mechanism. The strengths of a push mechanism compensate for the weaknesses of the pull mechanism, and vice versa. For example, a strength of a purely pull-based scheme such as NTP is that it gives maximum control to individual sensor nodes, who can request on-demand synchronization at any time. However, pull-based schemes invariably incur high overhead as each sensor node attempts to individually synchronize itself with the network. Pull-based schemes have also suffered from lower accuracy, due to wide variation in propagation delays due to wireless collisions [25]. A strength of a purely push-based mechanism is that it gives control to reference nodes, e.g. base stations, and allows for a low overhead method of quickly synchronizing large portions of the network. However, push-based schemes are vulnerable to lost packets, which would leave downstream sensor node(s) in an unsynchronized state. Such unsynchronized nodes are forced to wait until the next synchronization period. Prior proposals for time synchronization in sensor networks have at most focused on a single mechanism, and therefore suffer from the weaknesses of that particular mechanism.

Our integrated bidirectional approach gives full flexibility to both the base station and individual sensor nodes. The push-based mechanism is used as a lightweight synchronization mechanism for most sensor nodes. In case such a synchronization message is lost due to wireless collisions or fading, then the sensor nodes have the flexibility to initiate or request synchronization. Flexibility is enhanced by permitting both push and pull-based mechanisms to be parameterized, e.g. by their frequency of occurrence or multihop range.

II.B. Definitions

TSync is flexible and self-organized. Neither a fixed topology nor the guarantee of delivery latency is required in order to deploy the TSync service in a WSN. A physical broadcast channel is required, which is automatically satisfied by the wireless medium. A connected network is also required in order to spread the synchronization ripple to nodes network wide.

The TSync service assumes the coexistence of reference nodes and normal sensor nodes in a WSN. A **"reference node"** periodically transmits beacon messages to its neighbors. These beacon messages initiate the synchronization waves. Multiple reference nodes are allowed to operate in the system simultaneously. A sensor node in TSync will dynamically select the nearest reference node as its reference for clock synchronization.

TSync exploits the usage of multi-channel radios to improve precision, minimize the communication overhead and lower the energy consumption. A common **control channel** is shared by all the sensor nodes for delivery of beacon messages and control packets. This control channel can be the same one as is used for general data traffic. Each sensor node is also assigned a unique **clock channel** different from all its neighbors' clock channels. Usage of a dedicated clock channel reduces the variation in propagation delay caused by packet collisions and retransmissions, thereby improving the accuracy of clock estimation. As we observe in section VII, it is possible to deploy TSync in a WSN with only single-frequency radios, i.e. no dedicated clock channel, though accuracy will suffer.

An explanation of a standard two-way message exchange between a pair of nodes employing SNTP for synchronization is helpful to understand TSync's synchronization design. As illustrated in Figure 2, node A and node B wish to synchronize with each other. Node A sends a message to node B who then sends a reply message back to node A. t1 and t4 are measured by A's local clock while t2 and t3 represent the local clock's readings on B. We define the message propagation delay to be d1 and the local clock offset between A and B to be d2. The propagation delay d1 and the clock offset is assumed to be constant during the message exchange between A and B.

Node A initiates the synchronization by sending B a packet at time t1. The value t1, i.e. the original timestamp, is also contained in this packet. At time t2, Node B receives this packet. Here, t2 is called the received timestamp, and

$$t2 = t1 + d1 + d2 \tag{1}$$

At time t3, Node B acknowledge A's synchronization effort by sending back an ack packet which includes t2 and t3. Node A receives this packet at time t4, and

$$t4 = t3 + d1 - d2 \tag{2}$$

Thus, d1 and d2 can be estimated with the following simple formula:

$$d1 = \frac{(t2 - t1) + (t4 - t3)}{2} \tag{3}$$

$$d2 = \frac{(t2 - t1) - (t4 - t3)}{2} \tag{4}$$

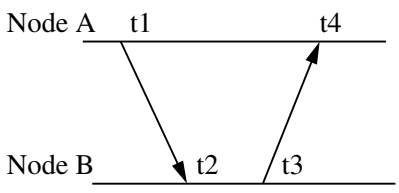

Figure 2: Synchronization algorithm timeline.

As a result, the clock on node A could be synchronized with node B's clock by adding offset d2. This procedure synchronizes the sender to the receiver, e.g. A to B, and can be applied in reverse as well.

Traditional synchronization protocols such as SNTP above assume that the delay d1 is the same in both directions. In reality, variation in the forward and reverse delays introduces errors that limit the accuracy of such time synchronization algorithms. Major sources of delay during time synchronization have been categorized into four distinct components, namely the send time, access time, propagation time and the receive time [10]. The *send time* is affected by the operating system overhead, such as context switching and resource allocation during construction of the message. Timing error is minimized by obtaining the timestamp at as low a level as possible. *Access time* is the delay that occurs when the node tries to access the medium. The MAC layer protocol determines this access time. The send time and the access time together contribute to errors in the estimation of t1 and t3 in the above example. *Propagation time* is the time needed for the message to be delivered from the sender to the receiver. For multi-hop time synchronization, this is a major error source. The propagation delay is nearly constant in one broadcasting domain and is only related to the speed that the message is tranmitted on the media. However, propagation delay varies significantly in multi-hop wide area networks due to random factors such as the backoff time after collisions, retransmission and queuing delays. This contributes to variations in d1, which violates the initial assumption of constancy. Finally, the *receive time* is the delay between the time when the message arrives at the receiver's radio interface and the time when the system is notifed about the arrival. Operating system processing needed to generate the message notification signal will affect the *receive time* and thereby the precision of estimating t2 and t4.

III. Push: HRTS - Hierarchy Referencing Time Synchronization Protocol

The first component of TSync's bidirectional time synchronization service is the push-based Hierarchy Referencing Time Synchronization (HRTS) Protocol. The goal of HRTS is to enable central authorities to synchronize the vast majority of a WSN in a lightweight manner. Protocol specifics based on a single reference node are discussed first, followed by an analysis, and then a generalization of the protocol to multiple reference nodes.

III.A. Single Reference Node

As shown in Figure 3, HRTS consists of three simple steps that are repeated at each level in the hierarchy. First, a base station, namely the reference node, broadcasts a beacon on the control channel (Figure 3(a)). One child node specified by the reference node will jump to the specified clock channel, and will send a reply on the clock channel (Figure 3(b)). The base station will then calculate the clock offset and broadcast it to all child nodes, synchronizing the first ripple of child nodes around the base station (Figure 3(c)). This process can be repeated at subsequent levels in the hierarchy further from the base station (Figure 3(d)).

The HRTS protocol is explained in more detail as follows:

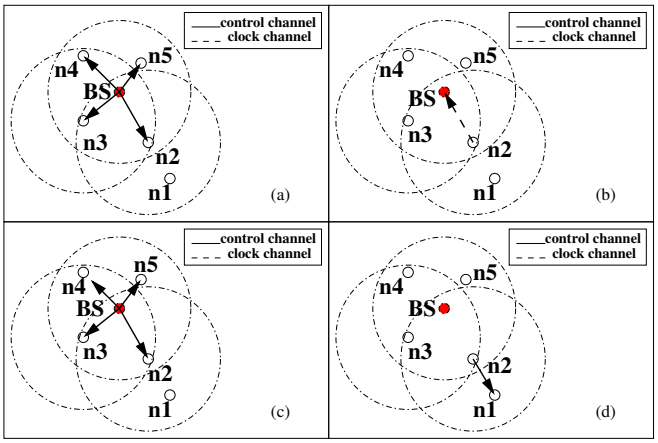

Figure 3: Push-based time synchronization: (a) Reference node broadcasts (b) A neighbor replies (c) All neighbors are synchronized (d) Repeat at lower layers

Step 1: BS initiates the sync_begin announcement with time t1 using the control channel and then jumps to the clock channel. All interested nodes record the received time of the announcement. BS randomly specifies one of its children, e.g. n2, in the announcement. The node n2 jumps to the specified clock channel. (Figure 3(a))

Step 2: At time t3, n2 replies to the BS with its received times t2 and t3. (Figure 3(b))

Step 3: BS now owns all time stamps from t1 to t4. It calculates d2 and then broadcasts t2,d2 on the control channel.(Figure 3(c))

Step 4: All interested neighbors, e.g. n2, n3, n4 and n5, compare the time t2 with their received timestamp t2'. For example, n3 calculates the offset d' as

$$d' = t2 - t2' \tag{5}$$

Finally, the time on n3 is corrected as:

$$T = t + d2 + d' \tag{6}$$

t is the local clock reading.

Step 5: n2, n3, n4 and n5 initiate the sync_begin to their downstream nodes. (Figure 3(d))

We assume that each sensor node knows about its neighbors when it initiates the synchronization process. In Step 1, the response node is specified in the announcement. It's the only node that jumps to the clock channel specified by the BS. The other nodes are not disturbed by the synchronization conversation between BS and n2 and can conduct normal data communication while waiting for the second update from the BS, the reference node. A timer is set in the BS when the announcement is transmitted. In case the

announcement is lost on its way to n2, the BS goes back to the normal control channel after the timer expires and thus avoids indefinite waiting.

As the synchronization ripple spreads from the reference node to the rest of the network, a multi-level hierarchy is dynamically constructed. Levels are assigned to each node based on its distance to base, i.e. # hops to the central reference point. Initiated from the reference nodes, the synchronization steps described above are repeated by the nodes on each level from the base to the leaves. To avoid being updated by redundant synchronization requests from peers or downstream nodes, HRTS allows the nodes to parameterize their requests with two variables, i.e. "level" and "depth":

level A "level" indicating the number of hops to the synchronization point is contained in each sync_begin packet. At the very beginning of each synchronization ripple, the reference nodes set the level to 0 in the sync_begin packet. If a node M is updated by a sync_begin packet marked by level n, it will set its level to n+1 and then broadcast the sync_begin message to all its neighbors with level n+1. If M receives another sync_begin packet later during the same synchronization period, M will look into the "level" contained in this packet. If the new level is equal to or larger than n, then M will just ignore this updating request. Otherwise, it will respond to this sync_begin packet and update itself based on the sender. Following this procedure, a tree is constructed dynamically with the reference nodes sitting at the base. Each node is associated with a level according to its distance to the base. For example, in Figure 3, the BS is at level 0, while all its neighbors n2, n3, n4, and n5 are at level 1 after being synchronized with the BS. After being updated by BS, the nodes n2, n3, n4 and n5 initiate the sync_begin packet to their child nodes. However, n2 and n3 are in each other's broadcasting domain. n3 will find n2 to be at the same level, and therefore will simply ignore the synchronization request from n2.

depth Besides "level", HRTS also allows network nodes to specify the radius of the synchronization ripple. The nodes could parameterize the request with a second field called "depth" in the sync_begin message. The initiating nodes set the "depth" field in the sync_begin packets. The value of "depth" is decremented by one in each level. The synchronization ripple stops spreading when the depth becomes zero. However, the downstream nodes could adjust the "depth" field according to their needs. With this flexible mechanism, the reference point will have the ability to control the range of the nodes that are updated.

III.B. Analysis of HRTS

The HRTS protocol exploits the broadcast nature of the wireless medium to establish a single common point of reference, i.e. the arrival time of the broadcast message is the same on all neighbor peers. This common reference point can be used to achieve synchronization in Step 4 of the previous subsection, i.e. t2 at node n2 occurred at the same instant as t2' at node n3. As the BS is synchronizing itself with n2, the other neighboring nodes can overhear the BS's initial broadcast to n2 as well as the BS's final update informing n2 of its offset d2. If in addition the BS includes the time t2 in the update sent to n2 (redundant for n2), then that allows all neighbors to synchronize. The intuition is that, since t2 and t2' occurred at the same instant, then overhearing t2 gives n3 its offset from n2's local clock and overhearing d2 gives n3 the offset from n2's local clock to the BS reference clock. Thus, n3 and all children of the BS can calculate their own offsets to the BS reference clock with only three messages (2 control broadcasts and 1 clock channel reply)!

HRTS is highly scalable and lightweight, since there is only one lightweight overhead exchange per hop between a parent node and all of its children. In contrast, synchronization in RBS happens between a pair of neighbors, which is called pair verification, rather than between a central node and all of its neighbors. As a result, RBS is susceptible to high overhead as the number of peers increases. The HRTS approach eliminates the potential broadcast storm that arises from pairwise verification, while at the same time preserving the advantage of reference broadcasting, namely the common reference point. Also, since the HRTS parent provides the reference broadcast that is heard by all children, then HRTS avoids the problem in RBS when two neighbors of an initiating peer are "hidden" from each other. The parameters used in the protocol dynamically assign the hierarchy level to each node during the spread of the synchronization ripple. No extra routing protocol is required. HRTS is lightweight since the number of required broadcasting messages is constant in one broadcasting domain. Only three broadcast messages are necessary for one broadcasting domain, no matter how dense the sensor nodes are.

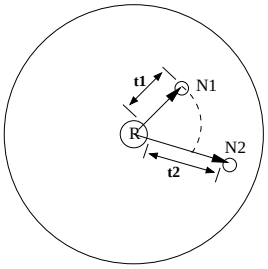

Figure 4: Variation in propagation delay within a single broadcast domain

In HRTS, the sender error is eliminated by comparing the received time on each node. The major error sources come from:

- *variance in the propagation delay* HRTS ignores the difference between the propagation time to different neighbors. As is illustrated in Figure 4, node R broadcasts to its neighbors n1 and n2. The propagation time needed for the message to arrive at n1 and n2 are t1 and t2, which are different in reality. As the propagation speed of electromagnetic signals through air is close to the speed of light, then for sensor network neighbors that are within tens of feet, the propaga-

tion time is in the nanosecond level and thus could be ignored compared with the other error sources. Also, HRTS makes the assumption that the propagation speeds are the same in different channels.

- *receiver error* Latency due to processing in the receiver side is attributable to operating system jitter. However, sensor operating systems can be designed so that this jitter becomes relatively small and deterministic, e.g. the time can be read immediately at each interrupt generated by the radio interface when a packet is received.

HRTS' current policy for selecting the child node to respond to the sync_begin message is a random selection. However, it is possible to incorporate historical knowledge from previous HRTS cycles in the selection of the next child responder. Moreover, previous HRTS responses may be combined to broadcast a composite value in Step 3. This may be useful to account for propagation delay differences between neighbors within a local broadcast domain, which we have assumed to be small, but which may become more relevant when the distances between neighbors becomes very large in a highly distributed WSN.

III.C. Multiple Reference Nodes

By parameterizing each synchronization request, HRTS permits the existence of multiple reference nodes in the sensor network to provide accurate clock readings.

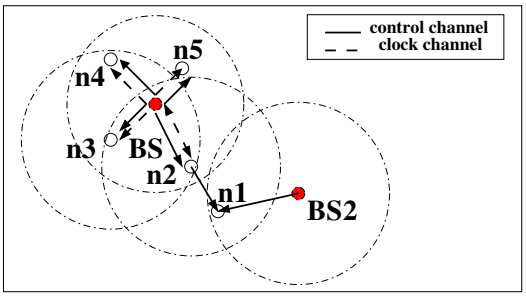

Figure 5: Synchronized by Multiple Reference Points

The existence of the "level" field in the sync_begin packet also serves the purpose of eliminating redundant updating requests from multiple reference nodes. By looking into the field of "level", a sensor node is able to recognize whether the message is from the nearer reference node or not. For example, in Figure 5, two reference nodes BS and BS2 exist in the sensor network. Node n1 is in the broadcasting domain of BS2 and 2 hops away from the BS. If the node n1 is updated by n2 before receiving a synchronization message from BS2, it will synchronize its clock again in response to a sync_begin packet from BS2, because the "level" in the packet from BS2 is 0, which is smaller than that from n2. On the contrary, n1 will ignore n2's updating request if it is updated first by BS2. When there are several reference nodes existing in the network together, a shortest path tree is formed around each reference node. This scheme allows each sensor node to select the nearest reference node in order to correct their local reading, thus minimizing jitter. When there are several reference nodes of the same distance to a sensor node, the node always selects the first arrival.

IV. Pull: ITR - Individual-based Time Request Protocol

The second component of TSync's bidirectional time synchronization service is the pull-based Individual Time Request (ITR) Protocol. ITR is designed to allow each sensor node to independently obtain the time or synchronize itself to its surrounding environment on-demand. ITR is proposed as a complementary mechanism to HRTS in order to give full flexibility to both individual nodes as well as central authorities, i.e. base stations. In some cases, HRTS' push-based ripple scheme may unnecessarily synchronize too many nodes despite being parameterized to a small radius, whereas ITR would be able to offer a targeted on-demand alternative to synchronize just those nodes that need it.

IV.A. Protocol Description

The ITR protocol is based on SNTP but integrates multichannel support to address the vulnerability of SNTP to variations in delay. It's well known that the accuracy of SNTP's time synchronization's algorithm is highly dependent on the variation in delay experienced during the roundtrip SNTP query. Indeed, the research literature has statistically modeled this variation [17]. MAC-layer collisions followed by hop-by-hop retransmissions in a wireless sensor network increase the variation in delay experienced by SNTP queries, and hence reduce the accuracy of SNTP time estimation. By employing a separate clock channel for time synchronization queries and responses, variations in latency due to collisions are reduced, resulting in more accurate clock estimation.

The ITR protocol is detailed as follows:

Step 1: Node n1 sends an ITR_QUERY on the control channel. The clock channel for transmitting the actual synchronization request is specified in the ITR_QUERY. (Figure. 6(a))

Step 2: n1's parent n2 hears the request. It then puts an ITR_ACK on the control channel to notify its parent, i.e. BS. In general, upstream parents continue repeating ITR_ACK's until a reference node is reached. (Figure. 6(b))

Step 3.a: n2's parent BS hears the ITR_ACK. BS then switches to the specified clock channel in the ITR_ACK for the incoming request. All nodes along the path are now switched to the specified clock channel.

Step 3.b: n2 receives the actual ITR synchronization request from n1 at the clock channel and forwards it to BS at the same clock channel as well. (Figure. 6(c))

Step 4, 5 and 6: The BS initiates the same procedure to send the time back to n1. (Figure. 6(d))

End: n1 synchronizes itself according to BS's feedback.

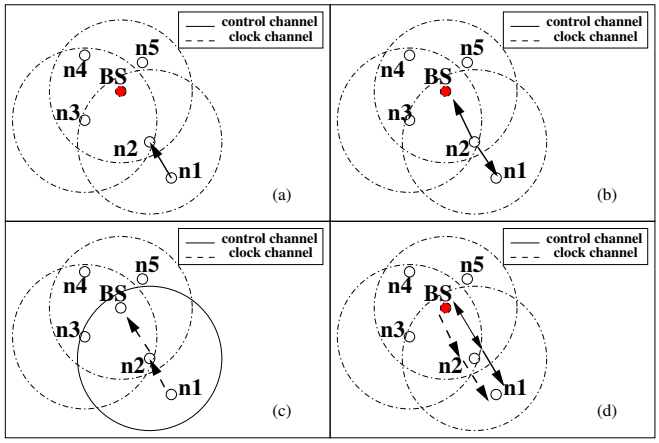

Figure 6: Pull-based multi-channel time synchronization avoids collisions and lowers delay variance (a)-(d)

The algorithm of ITR is illustrated in Figure 6. Similar to HRTS, ITR also allows the sensor nodes to parameterize their request. The "depth" is used here to specify the diameter of the query ripple spreading to its neighbors. By setting the "depth" field to different values, the sensor nodes can choose either to synchronize with a remote reference node many hops away or simply with its surrounding neighbors.

In the case of multiple hops to the BS, the ITR_ACK will propagate upwards through its parents until a reference node is encountered or the 'depth' field has expired. The path from requesting node to reference node (BS) will be "paved" using the same clock channel. If there are N immediate neighbors to a node requesting ITR synchronization, then up to N ITR_ACKs will be unicast outwards towards possibly N reference nodes. The first reference node that responds will be selected for synchronization. In the case that the topology is known in advance, i.e. by listening to HRTS messages, then the ITR_QUERY can be targeted to only one of the N neighbors, thereby limiting ITR_ACK propagation.

To synchronize to an unknown reference node, the sensor node simply sets the "depth" field in the request to infinity. This request will then be forwarded until the request either encounters a reference node or reaches the edge of the network. Such a request could be expensive and nodes may have to wait for a long time before the response is received. A timeout is therefore set when the request is first sent. When the timeout expires, the sensor node will think there is no reference node nearby and can then simply synchronize to the neighbor node that responds first.

If the depth field in ITR is set to be 1, then each node in ITR only makes an SNTP-like query to its upstream parent. In this way, clocks are distributed in a scalable manner. The query is only local to an upstream neighbor, rather than going all the way back to the reference node. This particular form of ITR is thus configured in a manner similar to the hierarchical SNTP approach of [25].

IV.B. Analysis of ITR

ITR is intended for use by nodes that wish to synchronize their clocks during the interval between two synchronization ripples pushed by HRTS. The majority of the sensor nodes are intended to be synchronized via HRTS' push mechanism.

Similar to SNTP, ITR is vulnerable to variations in the propagation delay in both communication directions. The receiver delay also contributes to the error in delay estimation. However, as we will show, ITR's multi-channel approach eliminates a large part of the variation in the transmission delay over a multi-hop network, as no other traffic exists on the same clock channel. The ITR_ACK message is also designed to be much smaller than the regular timestamping packets. Thus, the collision chances are reduced, especially when there is heavy traffic present. The resulting improvement in precision is demonstrated in Section VI.

While an intermediate node along the ITR route is busy servicing an ITR request, it ignores other ITR_ACK's. This minimizes the cross-traffic hence delay jitter experienced by the ITR synchronization packet. An additional consequence of this policy of dedicating a node to service a single ITR request is that the node may also ignore on-going data/control traffic through the node. Our assumption is that on-demand ITR synchronization will be invoked relatively infrequently and over a localized enough path such that the disruption to the rest of the sensor network will be relatively minor. Moreover, the urgent time-delay requirements of ITR packet traffic motivated our design choice of prioritizing service to time synchronization packets.

V. GPS-enabled Evaluation Framework

In order to evaluate the effectiveness of TSync in a live WSN, we developed a GPS-based framework for evaluating time-synchronization algorithms in-situ. This framework is based on the MANTIS nymph platform [9], which provides integrated GPS support in a small low-cost lightweight form factor. Other sensor platforms also support GPS, e.g. Zebranet, a system that is designed to have GPS, flash memory, CPU and two wireless radios working together to track wild animals [21].

The Lassen SQ GPS chip from Trimble is currently used with the MANTIS nymph sensor node, as illustrated in Figure 7. This integrated GPS chip provides the nymph with a pure clock, which has a precision within 200 nanoseconds. This clock is used to assess the accuracy of various time synchronization algorithms at each node in a WSN, providing a powerful framework to assess the accuracy of in-situ time synchronization algorithms down to the microsecond level over multi-hop wireless networks.

All nymphs in the experiments are connected to the GPS chip via a serial port. The PPS(pulse per second) pin on the GPS chip is connected to the nymph's external inter-

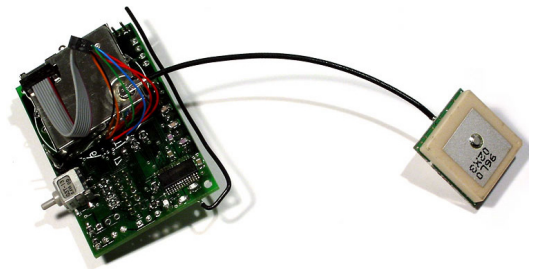

Figure 7: GPS-enabled sensor nodes using the MANTIS Nymph hardware platform

rupt pin. The GPS clock reading can be queried over the serial port. A local clock generated by the nymph's crystal is running on each nymph sensor node. As the crystals on different sensor nodes have different frequencies, each local clock will drift at a different rate. We thus first measure the individual clock drifts by comparing the local clocks to the GPS clock.

VI. Experiments and Performance Analysis

In this section, we verify the performance of TSync's services via an implementation on live nymph sensor nodes within our in-situ GPS-enabled evaluation framework [9]. We first measure the clock drift on each node and then use this drift to correct the clock reading. The performance of TSync is then evaluated in terms of its accuracy, overhead and scalability.

The two components of TSync, namely ITR and HRTS, are implemented independently as two different protocols in order to evaluate their individual performances. In ITR, the "depth" is set to infinity, so that the sensor nodes will search for the nearest reference node. HRTS is also parameterized to spread the synchronization ripple to all sensor nodes. Several algorithms are implemented for comparison, including SNTP and RBS [14]. For RBS, the gateway nodes are statically assigned, as well as the timestamp-converting path of chained gateways.

VI.A. Experimental Setup

The experimental setup is illustrated in Figure 8. The experiment consists of five deployed GPS-equipped sensor nymphs n1, n2, n3, n4 and BS (sensor node functioning as a base station), with the outermost node n1 requesting time synchronization over a three-hop network from the node BS. The Chipcon CC1000 radio on the MANTIS nymphs supports spread spectrum multi-channel communication, which enables implementation of TSync. To model modest packet traffic consisting of sensor data that could interfere with packets involved in time synchronization, a sixth node n5 is placed neighboring n3, n2 and BS. For each time synchronization algorithm, n5 is set to inject about 200 packets into the network within every 10 minute trial period. The injected packet size was varied from 20 bytes to 128 bytes.

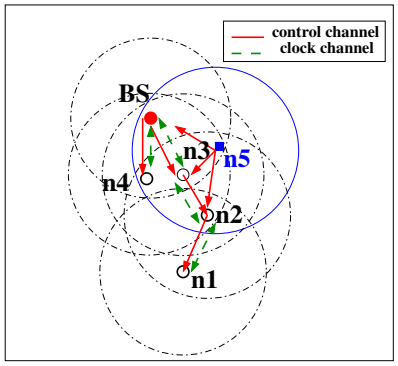

Figure 8: Experimental Setup

The clock reading should be taken as close as possible to the point when the event happened in order to minimize the sender error and the receiver error, as described earlier. The timestamps are processed at the lowest level of the MANTIS sensor OS, i.e. the interrupt handler of the radio interface. During wireless communication, each packet is appended with a preamble and a synchronization word at the head of the real packet in order to achieve DC balance and indicate the starting point to the receiver. When a packet is transmitted, the clock is read just prior to sending the first byte of the synchronization word. When a packet is received, the time when the synchronization word arrives is recorded immediately.

VI.B. Performance Analysis

VI.B.1. Clock Behavior

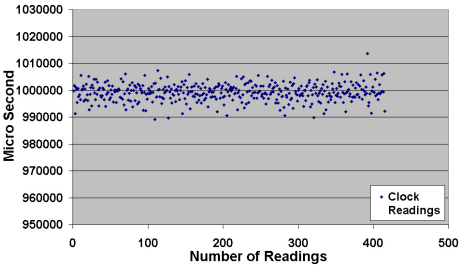

Figure 9: Clock Readings

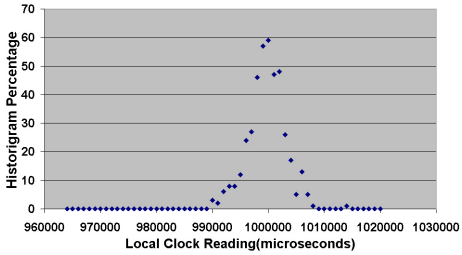

Figure 10: Clock Reading Distribution

Before analyzing the time synchronization algorithms,

our first goal was to isolate and characterize the behavior of the sensor clocks using our GPS-enabled sensor nodes. At the beginning of each experiment that measures the clock drift, all nodes are synchronized by a pulse to the same start point. The pulse is generated once per second. As soon as the pulse is received by the nymph, an interrupt is generated in a very tight loop and the local clock is compared to the GPS clock to see if exactly one second has elapsed on the local clock since the previous time. Typically, the local elapsed time, which we term the clock drift, is distributed around the mean of one second, with individual measurements slightly above and below the mean. The measurements are taken after the clock is stabilized. All nodes are measured within similar environmental conditions, e.g. same temperature.

For each sensor node, 5 different trials have been taken with more than 400 observations per trial in order to assess the clock's behavior. Figure 9 shows a sample set of clock drift observations taken in one 400-sample trial for one sensor. The clock drifts appear to be randomly distributed around the mean of about one second.

A closer statistical analysis of this single trace of data reveals that the clock readings form a roughly normal distribution with the mean value at 1,000,009 μsec, and standard deviation of 15.3 μsec, as shown in Figure 10. The standard deviation is larger than expected due to the long tails on either side of the curve. Though this particular sensor clock was found to drift on average 9 μsec fast for each second, other sensor clocks were found to drift on average slower than true time. However, the common property of all the sensor clocks that we measured, fast or slow, was that the error distribution formed a roughly normal distribution.

This analysis of clock behavior is used to correct for clock drift introduced by individual sensor nodes, as seen in the next subsection.

VI.B.2. Synchronization Accuracy

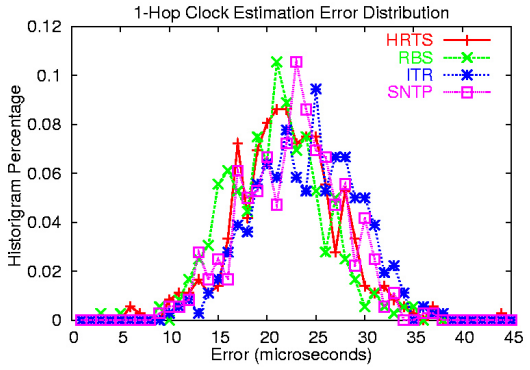

Figure 11: Single Hop Accuracy Comparison

A key metric for TSync is its accuracy as a time synchronization algorithm. We implemented SNTP, RBS, and TSync's HRTS and ITR over our experimental testbed of GPS-enabled nymphs. We executed each algorithm every 10 seconds, and compared the clock of a sensor node after

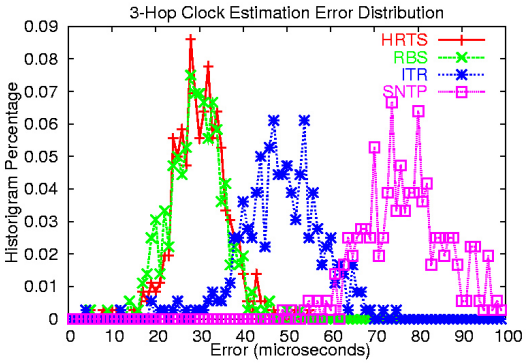

Figure 12: 3 Hop Accuracy Comparison

synchronization to the true GPS clock attached to that sensor node. For the case of three hops, the designated sensor node for evaluation was chosen to be n1. This comparison was repeated every 10 seconds during each one-hour trial for each algorithm. An initial random offset within 10 seconds is independently chosen by each sensor node in order to emulate the variations at boot up time.

The resulting distribution of errors was collected from all one-hour trials to produce Figure 11 and Figure 12, which show the distribution of the error in the accuracy of clock estimation for each of the four time synchronization algorithms over one hop and three hops respectively. The error distribution of all techniques appear to be roughly Gaussian. Table VI.B.2 and Table VI.B.2 present the mean in μseconds and variances, obtained by using the approximation method of numerical analysis.

The error values in both the figures and table for all algorithms have been corrected for the individual clock drifts at nodes n1, n2, etc., obtained from the preceding section's analysis. In the absence of such correction for clock drift, TSync will continue to function properly, though with less accuracy. This same limitation applies to the other algorithms as well. Correction for clock drift requires that each node's clock behavior be characterized a priori, that this characterization remain relevant over time, and that this characterization be made known to the time synchronization algorithm. The correction value is calculated by following an approach similar to [14], obtaining a least-squares linear regression estimate of the clock drift given statistics as from Figure 10. This offers a fast and convenient method for finding the best fit line through the error observed each second. A detailed study of clock skew exceeds the scope of this paper. The correction value is added or subtracted from a node's local time in order to correct for clock drift. As a result, the analysis of Figure 11 and Figure 12 can focus just on the inaccuracy introduced by the time synchronization algorithms themselves.

In this context, our experimental results in Figure 11 reveal that over a single hop, all algorithms achieve roughly similar accuracy. However, as the number of hops increases, our experimental results in Figure 12 over 3 hops reveal a significant increase in disparity between the accuracy achieved by pull and push-based algorithms. SNTP

Table 1: Approximate Mean and Variances of Single Hop Clock Estimation Error

Protocol	mean	variance
SNTP	22.3857	24.7843
ITR	23.6871	25.3742
RBS	20.3765	22.4728
HRTS	21.2342	23.4786

Table 2: Approximate Mean and Variances of 3-Hop Clock Estimation Error

Protocol	mean	variance
SNTP	75.9573	96.5837
ITR	48.4831	62.8639
RBS	28.8974	31.4765
HRTS	29.4762	32.0584

performs with the least accuracy. SNTP's inaccuracy can be attributed to its sensitivity to variations in propagation delay caused by congestion as well as packet interference losses in both directions. The analysis of ITR's pull-based mechanism reveals that ITR's multi-channel enhancements to SNTP significantly reduce delay varation, reducing the error in time synchronization precision by roughly half. However, ITR's pull-based mechanism still suffers inherently from variations in multi-hop queueing over the WSN. Turning to the push-based broadcast techniques, both HRTS and RBS have very similar error distributions, and both are much improved over multi-channel ITR, with average errors of about 29 μsec being quite common. Push-based synchronization achieves greater accuracy because it exploits the reference broadcasting property of wireless links, allowing all the sensor nodes in the same broadcasting domain to synchronize to a single common reference point, namely the broadcast message. As a result, the propagation delay is limited to one broadcasting domain and thus there is no extra variation in delay, as would be experienced by the pull-based techniques. This contributes to minimizing the error between the local time and the absolute time.

The true advantage of HRTS compared to RBS is revealed in the next section by examining the comparative overhead of the two schemes.

VI.B.3. Overhead

In this section, we evaluate each algorithm in terms of its overhead, i.e. the total number of messages exchanged during each propagation step. We begin with a simple example of synchronizing two receivers given a parent node, and then discuss how the algorithms scale to dense networks with many receivers. First, RBS employs two messages if the two receivers can directly hear each other, namely one beacon message and one from the peer nodes. Otherwise, three messages are required, since the time conversion path needs to be constructed by the parent node. SNTP requires four messages as each individual receiver needs to acquire the time from the sender. HRTS requires three messages during synchronization. A conversation between the parent and one of the children requires two packets while the parent also needs to broadcast a final update. The above analysis doesn't consider repeated messages due to packet collision. The number of necessary messages for two receivers in one broadcasting domain is summarized for each algorithm in Table 3.

Table 3: Necessary Synchronization Messages for One Broadcasting Domain and Two Receivers

Protocol	Message Number
HRTS	3
RBS	2 if they could hear each other Otherwise at least 3
SNTP	4

However, as the density and number of receivers increases, the number of overhead messages required for synchronization can increase substantially. For example, the number of necessary messages for RBS increases dramatically due to pairwise verification. The construction of time-conversion gateway chain paths requires even more routing messages. In contrast, the number of necessary messages for HRTS grows slowly since the messaging overhead required in one domain is a constant and redundant messages are largely eliminated by the parameterization mechanism. The routing in HRTS also does not entail extra overhead as it is constructed naturally by leveraging the broadcast medium.

To assess messaging overhead in a multi-node sensor network, we evaluated TSync in an ns2 simulation scenario. The ns2 simulation is confined to assessing overhead alone, and is not used to measure the accuracy of time synchronization. We randomly placed 200 nodes in a 400x400 area. The nodes' positions are static. A node is randomly chosen to be the reference node during initialization. Network-wide time synchronization is initiated from this reference point by sending a sync_begin to surrounding neighbors. In the simulation, the clock channel and the control channel use the same channel.

Figure 13 shows the number of messages required to propagate one synchronization ripple throughout the network, i.e. the total number of messages necessary in order that the sensor nodes in each broadcasting domain are synchronized to each other. As the number of receivers increases, RBS's overhead increases significantly as more messages are required between pairs of nodes. Without the appropriate optimization, the number of messages could increase exponentially. In contrast, HRTS is considerably less sensitive to the density of the sensor network, since all nodes in any given broadcasting domain can be synchronized with three messages. This supports the contention that HRTS achieves many of the benefits of broadcast-

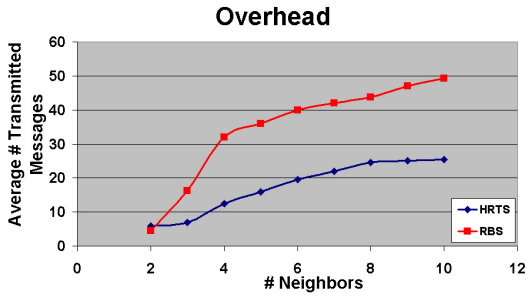

Figure 13: Overhead Comparison with 200 nodes. The x coordinate is the average number of average neighbors for each node.

based time synchronization while avoiding the pairwise overhead of RBS.

VII. Discussion and Future Work

TSync could be easily adapted to wireless sensor networks that don't support multiple channels. The clock synchronization messages would use the same channel as used for data communication. The accuracy would decrease due to increased packet collisions.

HRTS and ITR achieve different accuracies, so sensor nodes have the option of initiating coarse time synchronization on demand by using ITR, or of simply waiting for the next sync_begin cycle of HRTS initiated by a reference node. The network may choose to offer HRTS broadcast synchronization on a limited periodic basis, where the policy for periodicity could be application-specific and adaptive to the energy status of the sensor network.

An alternative to HRTS' scheme of pre-selecting the responding node from among all children is to have all the child nodes jump to the clock channel and employ a random backoff timer that selects the first child who grabs the channel as the respondent to the reference node. This scheme has the advantage of not requiring a priori knowledge of the neighborhood topology. However, the advantage of pre-selecting the responding node is that other interested but non-selected nodes can simply stay on the control/data channel, continuing to service other packets on this channel and eventually collecting from this channel the synchronization packet broadcast from the reference node. Unnecessary jumping to/from the clock channel is thus avoided.

An interesting issue concerns at what layer to place time synchronization. In our implementation, time synchronization packets were intercepted at or below the MAC layer to understand the limits to accuracy that can obtained by removing operating system jitter. In theory, TSync can be implemented at any layer, e.g. as an application layer thread in the MANTIS OS. This approach would be more portable, but would also introduce more imprecision. It remains as a future research issue whether application layer implementions of TSync and other time synchronization algorithms can provide sufficient clock accuracy.

Our future work will be focused on security and robustness issues of time synchronization, as well as in-situ deployment issues such as application layer implementation that we expect to experience for TSync. Security is important for time synchronization, since it's easy for a malicious node to completely destroy the precision of the TSync system by variably delaying packets on the clock channel. The current design requires mutual trust between all nodes and thus is vulnerable to malicious nodes. The energy consumption for the sensor network will also be a design consideration in later versions of TSync.

VIII. Conclusion

In this paper, we have introduced a light-weight bidirectional time synchronization service TSync for networked wireless sensors. TSync's comprehensive service consists of two components, namely a push-based HRTS protocol and a pull-based ITR protocol. Both approaches can be flexibly parameterized to suit the time synchronization needs of a given application. We show that our service can synchronize all the sensor nodes to within an average accuracy of around 21 μsec over a single hop and 29 μsec over three hops using push-based HRTS synchronization. The performance is comparable to reference broadcasting in terms of accuracy while the overhead of HRTS is far less than RBS. HRTS scales remarkably well because its number of messages is constant per broadcast domain. We also present results from a GPS-enabled framework for evaluating the accuracy of TSync in a live sensor network.

References

[1] A. Cerpa, J. Elson, D. Estrin, L. Girod, M. Hamilton, and J. Zhao: "Habitat Monitoring: Application Driver for Wireless Communications Technology", Proceedings of the First ACM SIGCOMM Workshop on Data Communications in Latin America, 2001.

[2] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, J. Anderson: "Wireless Sensor Networks for Habitat Monitoring", First ACM Workshop on Wireless Sensor Networks and Applications (WSNA) 2002, pp. 88-97.

[3] ARGUS Advanced Remote Ground Unattended Sensor Systems, Department of Defense, U.S. Air Force, http://www.globalsecurity.org/intell/systems/arguss.htm.

[4] C. Guo, L. C. Zhong and J. M. Rabaey, "Low Power Distributed MAC for Ad Hoc Sensor Radio Networks", Proceedings of IEEE GlobeCom 2001, San Antonio, November 25-29, 2001

[5] C. Liao, M. Maronosi, D. Clark: "Experience With an Adaptive Globally-Synchronizing Clock Algorithm", Eleventh Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA), 1999, pp. 106-114.

[6] D. Mills, Z. Yang, T. Marsland (Eds.): "Internet Time Synchronization: the Network Time Protocol Global States and Time in Distributed Systems", IEEE Computer Society Press 1991

[7] D. .Mills: ”Internet Time Synchronization: The Network Time Protocol”, Global States and Time in Distributed Systems. IEEE Computer Society Press, 1994.

[8] F. Martin, B. Mikhak, and B. Silverman: ”MetaCricket A designer’s kit for making computational devices”, IBM Systems Journal, vol. 39, nos. 3 and 4, 2000.

[9] H. Abrach, S. Bhatti, J. Carlson, H. Dai, J. Rose, A. Sheth, B. Shucker, R. Han, ”MANTIS: System Support for MultimodAl NeTworks of In-situ Sensors”, 2nd ACM International Workshop on Wireless Sensor Networks and Applications (WSNA) 2003.

[10] H. Kopetz and W. Schwabl: ”Global time in distributed real-time systems”. Technical Report 15/89, Technische Universitat Wien, Wien Austria, October 1989.

[11] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E.Cayirci: ”Wireless Sensor Networks: A Survey”. Computer Networks, 38(4): 393-422, March 2002.

[12] J. Elson and D. Estrin: ”Time synchronization for wireless sensor networks”. IPDPS Workshop on Parallel and Distributed Computing Issues in Wireless Networks and Mobile Computing Apr. 2001.

[13] J. Elson and K. Rmer: ”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.

[14] J. Elson, L. Girod and D. Estrin: ”Fine-Grained Network Time Synchronization using Reference Broadcasts”. Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI 2002), Boston, MA. December 2002

[15] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, K. Pister: ”System Architecture Directions for Networked Sensors”. Proceedings of Ninth International Conference on Architectural Support for porgramming Languages and Operating Systems, November 2000.

[16] J. Hill and D. Culler: A wireless embedded sensor architecture for system-level optimization. Technical report, U.C. Berkeley, 2001.

[17] K. Arvind: ”Probabilistic Clock Synchronization in Distributed Systems”, IEEE Trans. parallel and Distributed Systems, vol. 5, no. 5, pp. 475-487, May 1994.

[18] M. L. Sichitiu and C. Veerarittiphan: ”Simple, Accurate Time Synchronization for Wireless Sensor Networks”, Proc. of the IEEE Wireless Communications and Networking Conference (WCNC 2003), New Orleans, LA, March 2003.

[19] M. Mock, R. Frings, E. Nett, and S. Trikaliotis: Continuous clock synchronization in wireless real-time applications. In The 19th IEEE Symposium on Reliable Distributed Systems (SRDS 00), pages 125 133, Washington - Brussels - Tokyo, October 2000.

[20] P. Enge, P. Misra: ”Special Issue on Global Positioning System”, Proceedings of the IEEE, Volume 87, No. 1, pp. 3-15. January, 1999

[21] P. Juang et. al.: ”Energy-efficient Computing for Wildlife Tracking: Design Tradeoffs and Early Experiences with ZebraNet”, Proceedings of the 10th Intl Conference on Architectural Support for Programming Languages and Operating Systems, San Jose, CA, Oct 2002.

[22] P. Misra, P. Enge: Global Positioning System: Signals, Measurements, and Performance, Ganga-Jamuna Press, 2001.

[23] P. Ver1ssimo, L. Rodrigues, and A. Casimiro: Cesiumspray: a precise and accurate global time service for large-scale systems. Technical report. NAV-TR-97-0001, Universidade de Lisboa, 1997.

[24] R. Karp, J. Elson, D. Estrin, and S. Shenker: Optimal and Global Time Synchronization in Sensornets”, Technical Report CENS Technical Report 0012, Center for Embedded Networked Sensing, University of California, Los Angeles, April 2003.

[25] S. Ganeriwal, R. Kumar, M. B. Srivastava: ”Timing-sync Protocol for Sensor Networks”, ACM SenSys 2003.

[26] Y. Nakagawa, H. Uchiyama, H. Kokaji, S. Takahashi, M. Suzuki and M. Kanaya: ”Multi-channel adhoc wireless local area network”, 48th IEEE Vehicular Technology Conference, Ottawa, Ont., Canada, 18-21 May 1998.

[27] MICA2 Motes, http://www.xbow.com.

[28] The Cricket Indoor Location System http://nms.lcs.mit.edu/projects/cricket

[29] Simple Network Time Protocol, (SNTP) version 4. IETF RFC 2030