Lightweight Time Synchronization for Sensor Networks

Jana van Greunen
University of California, Berkeley
2108 Allston Way, Suite 200
Berkeley, CA 94704, USA
+1 510.666.3102
janavg@eecs.berkeley.edu

Jan Rabaey
University of California, Berkeley
2108 Allston Way, Suite 200
Berkeley, CA 94704, USA
+ 1 510.666.3102
jan@eecs.berkeley.edu

ABSTRACT
This paper presents lightweight tree-based synchronization (LTS) methods for sensor networks. First, a single-hop, pair-wise synchronization scheme is analyzed. This scheme requires the exchange of only three messages and has Gaussian error properties. The single-hop approach is extended to a centralized multi-hop synchronization method. Multi-hop synchronization consists of pair-wise synchronizations performed along the edges of a spanning tree. Multi-hop synchronization requires only n-1 pair-wise synchronizations for a network of n nodes. In addition, we show that the communication complexity and accuracy of multi-hop synchronization is a function of the construction and depth of the spanning tree; several spanning-tree construction algorithms are described. Further, the required refresh rate of multi-hop synchronization is shown as a function of clock drift and the accuracy of single-hop synchronization. Finally, a distributed multi-hop synchronization is presented where nodes keep track of their own clock drift and their synchronization accuracy. In this scheme, nodes initialize their own resynchronization as needed.

Categories and Subject Descriptors
C.3 [**Special-Purpose And Application-Based Systems**]: *Real-time and embedded systems, Signal processing systems*

General Terms: Algorithms, Performance, Reliability.

Keywords
Synchronization, Lightweight, Spanning tree, Multi-hop.

1. INTRODUCTION
Many applications of sensor networks depend on the time accuracy kept by nodes in the network. In such applications, events observed by a node are timestamped with the node's local time, which should be accurate to within the limits imposed by the application and the timescale of the events under observation. Notably, time accuracy to within fractions of seconds is often the maximum accuracy required by sensor network applications. Thus, it is often sufficient to use a relaxed or *lightweight* synchronization method in sensor networks.

This research was sponsored by the Defense Advanced Research Projects Agency (DARPA) under contract F33615-02-2-4005.

WSNA'03, September 19, 2003, San Diego, California, USA.

Sensor networks require a way for nodes to synchronize their *clocks* (i.e. local times) to a global time. Traditional synchronization algorithms have focused on minimizing the synchronization error and achieving maximum accuracy, without regard to the computation- and communication energy expended by the algorithm. In sensor networks, however, energy is a highly constrained resource. Thus, traditional synchronization algorithms are not adequate as they place a heavy burden on the network resources.

In this paper we argue that the communication and computation requirements of synchronization can be significantly reduced by taking advantage of the relaxed accuracy constraints. We introduce synchronization schemes that sacrifice accuracy by performing synchronization less frequently and between fewer nodes. The efficiency of these schemes can be adjusted to perform to the desired accuracy.

The work in this paper is principally motivated by the Pico-radio [1] project. Pico-radio is targeted mainly at environment monitoring applications such as temperature control, traffic monitoring, and surveillance. Pico-radio nodes are designed to be self-powered by scavenging energy from the environment and are extremely power constrained. Thus, minimizing energy and power consumption is crucial.

The *lightweight tree-based synchronization* (LTS) algorithms developed in this paper are designed to work with generic low-cost sensor nodes. The algorithms focus on minimizing overhead (energy) while being robust and self-configuring. In particular, the algorithms operate correctly in the presence of node failures, dynamically varying channels, and node mobility.

This paper is organized as follows: Section 2 presents an overview of existing synchronization algorithms for wired networks as well as related work in sensor networks. Section 3 describes and analyzes single-hop synchronization, which is then extended to multi-hop tree-based synchronization in Section 4. Section 5 and 6 discuss a centralized and distributed multi-hop synchronization scheme respectively. Sections 7 and 8 present future work and a summary.

2. RELATED WORK

2.1 General Synchronization Techniques
Synchronization is important in many systems, both wired and wireless, and a large number of time synchronization schemes exist. Well-known synchronization schemes include GPS [2] and the Network Time Protocol (NTP) [3]. In general, synchronization can be implemented in software or hardware, and the algorithms can be probabilistic, deterministic, or statistical. This section pro-

vides a brief overview of the main characteristics of existing time synchronization schemes.

The following discussion is based on a classification of synchronization algorithms by Anceaume and Puaut [4]. They partition time synchronization into three main components: resynchronization event detection, remote clock estimation, and clock correction. Resynchronization event detection identifies the time at which nodes have to resynchronize their clocks. This identification can be performed in two ways. The first technique is based on initially synchronized clocks that are resynchronized at a constant rate of kR where R is the duration of a single synchronization round and k is a real number larger than one to prevent overlap between the rounds. The second technique for resynchronization detection relies on a specific node to send an initiating message to every other node j in the system when kR time has passed. Once the message reaches node j, node j initiates its own synchronization. Thus the accuracy of the synchronization is dependent on the message latency. Our centralized scheme presented in Section 5 uses a variation of the latter technique and the distributed scheme presented in Section 6 utilizes the first technique.

The *remote clock estimation* component is used to determine the local time of another node in a network. Remote clock estimation is performed with one of two techniques: "time transmission" when the time of a remote clock is sent in a message, or "remote clock reading" when the delay bounds are unknown. Both these techniques involve the transmission of additional messages. These additional messages add unwanted communication overhead and thus are to be avoided in a lightweight synchronization algorithms. Instead, our algorithms estimate the maximum deviation of the remote clock by using a probabilistic bound on synchronization accuracy and the clock drift since the last synchronization.

The last component of time synchronization is *clock correction*, which is used to update the local time of a node when a resynchronization event has occurred (the first component) and information about remote clocks has been estimated (the second component). Clock correction estimates an adjustment A_{pi} based on the "estimation set" [4]. The estimation set is a set containing relevant remote clock estimates which are produced by the clock estimation component. There is two broad classes of these clock correction functions: convergence average based functions (which return an average of the values contained in the estimation set) and convergence non-average based functions. The non-average techniques use only a subset of the estimation set to account for Byzantine failure.

Traditionally, synchronization techniques have focused on achieving maximum accuracy. In this sense our approach differs fundamentally because the objective is to minimize complexity (and therefore energy) of the synchronization algorithm. The accuracy is given as a constraint. The next section discusses a synchronization algorithm developed for sensor networks.

2.2 Related Work in Sensor Network Time Synchronization

To our knowledge, there are three distinct synchronization algorithms, Reference Broadcast (RBS [5]), TINY/MINI-SYNC [6], and Level synchronization [7]. In the first scheme (RBS), an intermediate node is used to synchronize the local time of two nodes. The intermediate node transmits a "reference packet" to the two nodes. The two nodes record the time that they received the packet and then exchange this recorded time to find the difference. The accuracy of RBS is mostly determined by the amount of time it takes either node to receive and process the reference packet; if there is a large difference in receiving time, the accuracy of RBS is reduced. RBS has a complexity of 4 received and 3 transmitted messages for two nodes. For n nodes and m reference broadcast packets, RBS has a complexity of $O(mn^2)$ – for each of the m received reference packets, a node exchanges information with all other $n-1$ receivers. A major goal of this paper is to develop an algorithm with much lower complexity.

In [5] the RBS algorithm is further extended to multi-hop (rather than pair-wise) using "multi-hop clock conversion." Multi-hop clock conversion is designed to synchronize at least two groups of nodes. A group of nodes is defined as all nodes that can receive reference packets from a particular broadcast node. Let A and B be two broadcast nodes. It is assumed that at least one receiver of node A is also a receiver of node B. Synchronization is then performed by finding the statistical best fit of the receiver time differences through the nodes in both groups. The multi-hop RBS algorithm relies on effective clustering of the nodes around the broadcast nodes. Mitra and Rabek [8] proposed an addition to RBS in the form of a clustering service that maintains a cluster cover of the sensor network. The service also ensures that there is overlap between the clusters so that inter-cluster synchronization can be efficiently performed. This clustering service adds considerable overhead to the RBS algorithm because the clustering topology needs to be constructed and maintained.

Elson et al. [9] proposes post-facto synchronization. Post-facto synchronization utilizes RBS synchronization, but nodes are synchronized only after a time-sensitive packet has been transmitted. If many such packets are transmitted, this scheme may result in many unnecessary pair-wise synchronizations.

The second type of synchronization TINY/MINI-SYNC [6] is based on the assumption that the nodes' clock drifts are of the following linear form: $t_i = a_i t + b_i$ where t_i is the local clock of node i, a_i and b_i are drift parameters, and t is "real" time. Under this assumption, the offset between two nodes is also linear. In order to perform pair-wise synchronization, TINY/MINI-SYNC nodes exchange time-stamped packets (timestamped in the same way described in the pair-wise synchronization algorithm in Section 3). These exchanged packets are used to estimate the best-fit offset line between the two nodes. As more packets are exchanged the computation complexity required for calculating the best-fit line increases. The linear constraint is used to identify redundant packets that are discarded to lower computation complexity. Each node performs this pair-wise synchronization scheme with each of its neighbors.

The third scheme [7], which utilizes level-based synchronization, introduces the pair-wise synchronization that we will use in this paper. This scheme was chosen because it is extremely simple and computationally efficient. The accuracy of this pair-wise synchronization scheme is determined by the sensor's radio characteristics. Due to the importance of pair-wise synchronization in the execution of our multi-hop synchronization scheme this pair-wise scheme will be presented and analyzed in detail in Section 3.

The multi-hop component of the level-based synchronization scheme differs from the scheme presented in this paper. In the level-based scheme each node is assigned a logical level indicat-

ing its distance from the chosen leader node. This level assignment is fixed for the lifetime of the leader. When new nodes join the synchronization they are required to initiate a "level discovery" phase. Pair-wise synchronizations are then performed between nodes in adjacent levels. The static nature of the level hierarchy reduces the robustness of this solution.

The next section presents a detailed overview of the technique for pair-wise time synchronization used in the remainder of the paper.

3. PAIR-WISE SYNCHRONIZATION

The following section describes a basic scheme to synchronize pairs of nodes. Nodes *j* and *k* can synchronize their local time by exchanging two packets with the following procedure:

- Node *j* transmits the first packet with a timestamp t_1 with respect to its local time.
- Node *k* records the time t_2 when it receives the first packet. Time t_2 is equal to t_1 plus the transmission time *D* from node *k* to *j* plus the offset *d* between node *j* and *k*'s clocks. Generally the transmission time *D* is unknown and is a function of the distance between the nodes and signal propagation characteristics.
- Next, node *k* transmits a second packet to *j* that contains t_1 and t_2. This packet is also timestamped by *k* at time t_3.
- Node *j* receives the second packet at time $t_4 = t_3 + D - d$. See Figure 1 for a graphical depiction of the exchange.

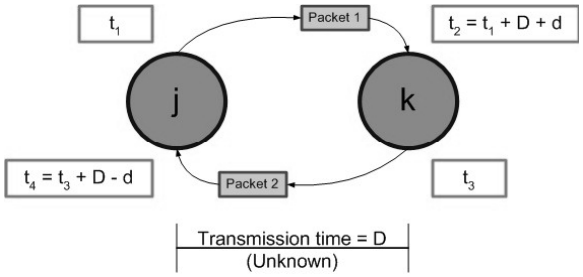

Figure 1: Packet exchange for pair-wise synchronization

- The offset *d* can be calculated at node *j* by subtracting t_4 from t_2.
  $\boldsymbol{t_2 - t_4 = t_1 - t_3 - D + D + 2d}$
  $\mathbf{d} = 0.5*(\boldsymbol{t_2 - t_4 - t_1 + t_3})$
- The two nodes are synchronized once node *j* has calculated the offset *d*. However, a third message is required if the offset *d* must also be communicated to node *k*.

The underlying assumption is that the transmission time is the same from *j* to *k* and *k* to *j,* that is $D_1 = D_2$. Of course D_1 and D_2 are not exactly equal and this introduces some error in the synchronization. In order to analyze this difference we provide a brief overview of the components that make up the transmission time *D* and the amount of error that each component contributes. Kopetz and Schwabl [10] have divided the transmission time into four parts:

- *Send time.* The time spent assembling the message at the sender, which includes processing and buffering time. The message is timestamped after the send time has completed so send time does not contributed to the difference in transmission times.
- *Propagation time.* The time for the signal to propagate across the physical medium between the two nodes. The propagation time is a function of the distance between the nodes. As the distances between nodes do not change rapidly, the propagation time is the same in both directions and thus does not contribute to the difference between transmission times.
- *Receive time.* The processing time required for the receiver to receive a message from the channel and notify the host of its arrival. Elson et al. [5] characterized the receive time using a testbed of "COTS MOTES," a narrowband radio and sensor platform developed by Warneke, Atwood, and Pister [11]. Results from receive delay differences showed that the distribution of inter-receiver variability was Gaussian with zero mean and a variance σ of 11.1 microseconds. The experiment validates the use of a Gaussian distribution to model the variability of receive time.
- *Access time.* The delay associated with accessing the channel, including carrier sensing. The differences in access times arise in much the same way that the difference in receive time do because the packets go through the same physical and MAC layers of the radio. Thus, as with receive time, we assume that differences in access times among nodes are also Gaussian with zero mean

When the error from the receive- and access time are combined, their variance increases at most four times (depending on the degree of correlation between them). Thus, when two nodes synchronize with each other, we can say with 99% confidence that the accuracy will be within $(2.3)*(4\cdot\sigma)$ or $9.2*\sigma$. For the COTS MOTES [11] this is an accuracy of 0.1 milliseconds.

In the next section we will extend the pair-wise (single-hop) synchronization algorithm introduced in this section to *multi-hop* synchronization. In addition to accuracy of the synchronization we will also discuss stability (drift of clocks over time) and a way to determine when nodes must resynchronize.

4. MULTI-HOP SYNCHRONIZATION

Multi-hop synchronization is an extension of the pair-wise synchronization algorithm. In a straightforward extension of pair-wise synchronization, a group of *n* nodes requires n^2 pair-wise synchronizations. Due to the relatively low accuracy requirements of our sensor network, we will avoid this n^2 factor by linearizing the synchronization by performing pair-wise synchronization only along network edges that form a spanning tree structure, described later.

There are several important considerations for multi-hop synchronization that influence the design of an efficient algorithm:

- *Global reference.* We assume that at least one node in the network has access to a global time reference. We will further assume that the global time kept by any reference node is orders of magnitude more accurate than the accuracy achievable by the single-hop synchronization. All nodes with reference to global time are stationary.
- *Selective synchronization.* Multi-hop synchronization can aim to keep all nodes synchronized at all times, or we can perform selective synchronization. In other words, the algorithm

can synchronize only the nodes that are transmitting time-sensitive data.

- *Resynchronization rate.* Due to clock drift, the nodes will periodically need to be resynchronized. Here we assume a clock drift is bounded. In [4] a node's clock denoted by $H(t)$ is defined to be "ρ - bounded" provided that for all real time t,

$$\frac{1}{(1+\rho)} \le \frac{dH(t)}{dt} \le 1+\rho$$

  A clock $H(t)$ that drifts at a constant rate will have $\rho = 1.0$. The resynchronization can be done synchronously (all the nodes at once) or asynchronously.

- *Error estimation & limitation.* The synchronization algorithm itself should keep track of accuracy performance and the errors produced by clock drift among nodes. When the algorithm determines that node clocks have (or might have) drifted, a resynchronization scheme should be invoked.

- *Robustness.* There should not be a single point of failure in the system (except maybe the reference node) and the algorithm should be robust to node failures.

- *Mobility.* Synchronization should work for both stationary or mobile nodes

- *Propagation of error.* When multi-hop synchronization is performed by the algorithm presented in this paper (i.e. along the edges of a spanning tree), the error will grow linearly as a function of the number of hops from the reference node. Thus, we want to limit the worst-case depth of the spanning tree. In our algorithm the leaf nodes of the spanning tree will have the greatest synchronization error but this error is bounded to within the required accuracy limits.

In the next sections we propose two algorithms for multi-hop synchronization. The first algorithm is a centralized approach in which the synchronization and periodic updates are generated from a reference node. The second scheme is a distributed multi-hop synchronization method where the nodes (and not the reference nodes) are responsible for initiating and performing resynchronization.

5. CENTRALIZED MULTI-HOP LTS

Centralized multi-hop synchronization is a simple linear extension of the single-hop synchronization. The basis of the algorithm is the construction (either offline or dynamic) of a low-depth spanning tree T comprising the nodes in the network. In general, a new spanning tree is constructed each time the algorithm is performed. In order to synchronize nodes in the tree, pair-wise synchronizations are performed along the edges of T. In *centralized* multi-hop synchronization, the reference node initiates the synchronization by synchronizing with all immediate (single-hop) children in T. Next, each child of the reference node synchronizes with their subsequent children. This process continues until the leaf nodes of T are reached. The algorithm terminates when all the leaf nodes have been synchronized. This running time of the algorithm is proportional to the depth of the tree.

5.1 Analysis of Error

The variance of the synchronization error increases along each branch of the tree as a linear function of the number of hops. This is because the errors resulting from the respective pair-wise synchronizations are independent and thus additive. As assumed in the pair-wise synchronization discussion, the synchronization error between two adjacent nodes is a Gaussian random variable with a variance of four times the receiver variance σ. Thus, for a node at depth d in the spanning tree, the expected error is zero but the variance of the error is $4*d*\sigma$.

The following example illustrates the effect of error accumulation on accuracy. Consider a network with 1000 randomly distributed nodes in a rectangular region of size 140m*140m with a radio range of 10m and a reference node in the center of the region. If the spanning tree is created by breadth-first search, the longest possible path in the network is along a diagonal of length $70*2^{½}$ which consists of at most 10 hops. Thus the maximum resulting variance at the leaf nodes will be 40σ. If the nodes are COTS MOTES then with 99% confidence the leaf nodes will be accurate to within $2.3*40\sigma$, which is an accuracy of 1ms.

The error accumulation is highly associated with the spanning tree used for synchronization, specifically its depth. In the next section we will consider algorithms for efficiently constructing low-depth spanning trees in a distributed manner.

5.2 Creating a Spanning Tree

We would like to construct a spanning tree that maximizes the synchronization accuracy. Thus, based on the previous discussion, an optimal tree is one with minimum depth. If we consider clock drift, the accuracy of synchronization is also affected by the running time of the algorithm. To minimize running time, the synchronization should occur in parallel along all branches so that all leaf nodes finish similar times. One type of tree construction that yields both these properties is breadth first search.

Breadth-first-search has a higher communication overhead compared to other tree-construction algorithms. The communication complexity (i.e. the number of messages generated) of breadth first search can be minimized to $10*n*m^{½}$ where n is the number of nodes and m is the number of edges between them. Breadth first search is also difficult to perform in a distributed manner.

In addition to breadth-first-search, there are two other tree-construction algorithms with desirable properties. Distributed depth first search (DDFS) developed by Awerbuch.[12] is a computationally efficient algorithm (see Figure 2 for a detailed version of the algorithm). Savings in communication arise because each node informs its neighbors when it is visited the first time, before it continues the recursive search among its children. Thus, DDFS eliminates the return calls along non-tree edges. The communication complexity of the algorithm is $4*m$ (where m is the number of edges) and the time complexity is bounded by $4n$-2.

```
Start the algorithm at node u the initiator:
visited_u := true ;
    for all x ϵ Neigh_u do send <visit> to x;
    for all x ϵ Neigh_u do receive <ack> from x;
    for some w ϵ Neigh_u do send <dfs> to w; statusu[w] := cal
    end


Upon receipt of <visit> from v:
    statusu[v] := done ; send <ack> to v

Upon receipt of <dfs> from v:
    if not visited_u then
      begin visited_u := true; status_u[v] := father;
          begin forall x ϵ Neigh_u \ {v} do send <visit> to x;
          forall x ϵ Neigh_u \ {v} do receive <ack> from x;
      end;
    if there is a w ϵ Neigh_u with status_u[w] = unused
      begin send <dfs> to w; status_u[w] := cal
    else if there is a w ϵ Neigh_u with status_u[w] = father
      begin send <dfs> to w end
    else (* initiator *) stop
```

Figure 2: Awerbuch's distributed depth-first search (algorithm description taken from [12])

A second tree-exploration algorithm is called the "Echo" algorithm, described in [12] (see Figure 3 for the algorithm). This algorithm is not computationally efficient, i.e. it actually has an O(*nm*) running time, but in practice it completes with O(*d*) running time where *d* is the depth of the tree. This short completion time will increase the accuracy of the synchronization.

```
var  rec_u       : integer init 0;
     father_u    : neighbor init undef;

Algorithm for the initiator:
    forall v ϵ Neigh_u do send <echo> to v ;
    while rec_u < |Neigh_u| do
      begin receive <echo> ; rec_u := rec_u + 1 end


Algorithm for other nodes:
    receive <echo> from w ; father_u := w ; rec_u := 1;
    forall v ϵ Neigh_u \ {w} do send <echo> to v ;
    while rec_u < |Neigh_u| do
      begin receive <echo> ; rec_u := rec_u + 1 end ;
    send <echo> to father_u
```

Figure 3:The Echo algorithm (algorithm description from [12])

5.3 Efficiency

The communication cost of the multi-hop synchronization algorithm arises from the spanning tree construction and the pair-wise synchronization along the tree's *n*-1 edges. Pair-wise synchronization has a fixed overhead of 3 messages per edge for a total of 3*n*-3 messages. The overhead for constructing the spanning tree depends on the complexity of the algorithm used to construct the tree. If DDFS is employed, the total overhead for centralized multi-hop synchronization is 3*n*-3 + 4**m* per network synchronization.

The network must periodically be resynchronized due to clock drift. The next section discusses finding the minimum allowable resynchronization rate while maintaining the accuracy required by the application.

5.4 Clock Drift and Resynchronization

It is desirable to keep all clocks accurate (with high probability) to within τ units of global time. In the centralized multi-hop algorithm, the reference node must periodically resynchronize the network. Two parameters are required by the reference node in order to calculate a good resynchronization interval: the instantaneous accuracy obtained by synchronizing the entire network, and the expected rate of clock drift.

In centralized multi-hop synchronization, the depth of the spanning tree determines the instantaneous accuracy of network synchronization. Each time the nodes are synchronized, the maximum depth of the spanning tree must be communicated to the reference node. This introduces the overhead of forwarding depth information back along the spanning tree when synchronization has completed. Given this maximum depth, a single synchronization session is accurate to within $9.2*d*\sigma$ (where σ is the variance per hop in units of time) with 99% probability.

By assuming a ρ-bounded clock, the expected clock drift rate will not exceed ρ. Thus, in order to maintain time accuracy to within τ units of global time, the reference node must resynchronize at a rate of at least $(\tau - 9.2*d*\sigma)/\rho$. (The numerator represents the amount of time that the clock can drift and the denominator represents the drift rate). In this paper, we assume that all nodes know their clock-drift ρ. This is a reasonable assumption because the clock drift of oscillators can be found in standard specification sheets and can easily be programmed on the nodes during assembly or during a network initialization phase. If we choose σ to be 11.1 microseconds as for the COTS nodes, the drift ρ to be the drift of a typical quartz crystal, which is 20-50 parts per million, a depth of 5 and an accuracy of 0.5 seconds, the resynchronization rate would be approximately 1mHz or once every 990 seconds. The reference node can calculate this rate and periodically generate a resynchronization.

Periodic resynchronization is complicated as it is possible to initiate a new synchronization session *B* before the previous session *A* is completed. The problem arises because a new spanning tree is created for each new synchronization session. It is possible for a node *k* to be synchronized with new information from session *B* but then resynchronized with stale information from session *A* due to differences in the spanning tree construction.

There are several ways to solve this problem and ensure that information from the most recent session is used. One way is to have the reference node include a monotonically increasing session number in the synchronization packets. Nodes along the tree edges could then discard synchronization packets from older sessions.

5.5 Robustness

The centralized multi-hop synchronization algorithm is robust in the following ways. First, although the algorithm is sensitive to failures in the reference node, backup or multiple reference nodes can be used. Second, given that a new spanning tree is created every time the network is synchronized, the algorithm is robust to

dynamic channel variations, changes in topology, changes in size, and node mobility. In particular, channel characteristics in sensor networks with mobile nodes are assumed to be constant relative to the time required to synchronize the network. The multi-hop algorithm can also keep the network synchronized to the required accuracy τ in the presence of network changes. This is because the reference node can calculate the maximum possible tree depth based on the radio range and network size and ensures that updates occur frequently enough to maintain accuracy.

6. DISTRIBUTED MULTI-HOP LTS

This algorithm performs node synchronization in a distributed fashion and does not make use of an overlay spanning tree to direct the pair-wise synchronizations. This algorithm also moves the resynchronization responsibility from the reference node to the nodes themselves. An individual node's resynchronization rate can be determined using the same parameters as the reference node uses in the centralized case. Therefore, to determine their resynchronization rates, nodes will need to obtain the following information: the desired accuracy τ, their distance d (in number of hops) from a reference node, their clock drift ρ, and a record of the time that has passed since they were synchronized. A particular node j needs to resynchronize at a rate of at least $(\tau - 9.2*d_j*\sigma)/\rho_j$. When a node j determines that it needs to be resynchronized, j will send a resynchronization request to the closest reference node. In order for j to resynchronize, all nodes along the routing path from the reference node to j will be synchronized in a pair-wise fashion.

If we assume that the clock drift ρ is the same for all nodes in the network, the nodes furthest from the reference node will have the greatest synchronization error and correspondingly the greatest synchronization rate. Therefore, the synchronization will be driven by these edge-nodes along paths that almost look like a reverse tree. An advantage of this algorithm is that certain nodes may not require frequent synchronization. If a node's rate of event observation is significantly lower than its required synchronization rate, it may not always need to be synchronized to the required accuracy. In other words, it is better to synchronize only when the node has a data packet to transmit. Thus, the nodes can opportunistically synchronize.

6.1 Avoiding cycles

When a synchronization request is forwarded from a leaf node to the reference node it is possible for a cycle to occur. A cycle occurs when the node at the head of the synchronization chain requests synchronization from a node that is lower down in the same request chain. Cycles occur because the routing is dynamic and a node requesting synchronization may not know the entire routing path at the time of the request. When cycles occur they cause deadlock, because the nodes mutually depend on each other for synchronization. The occurrence of cycles can be minimized by forwarding the known synchronization path from the child or requesting node to the new parent node at the time of a synchronization request. However, it is impossible to avoid cycles due to the asynchronous and distributed nature of the synchronization requests. Once a cycle has occurred a distributed graph-searching algorithm can be used to detect it. To our knowledge the searching algorithm with the lowest complexity (described in [13]) runs in $O(2m)$ where m is the number of edges in the sensor graph. This algorithm presents a considerable overhead.

We propose an alternative approach that does not rely on detecting cycles, but does avoid potential cycles. The approach works as follows: when a node sends a synchronization request to one of its neighbors it sets a timer that is proportional to its distance from the reference node. If the timer expires before a synchronization response from the neighbor arrives, the node simply initiates another synchronization request with a different neighbor. This scheme does not prevent cycles from occurring but reduces their impact at an overhead cost of additional synchronizations.

6.2 Algorithm Enhancements

Although some nodes have relaxed synchronization rates, in the distributed approach there are some potential inefficiencies of synchronization requests. For example, two adjacent nodes may attempt to resynchronize and send two separate synchronization requests. However, because the nodes are adjacent, it is more efficient to aggregate the requests. In general, duplicate requests can be eliminated by having each node keep track of pending requests from itself and other nodes. If a node k wishes to resynchronize or is forwarding a request from another node, it is beneficial for k to query each of its adjacent nodes to discover if any have pending requests. If so, k can forward the request to a node with pending requests, which aggregates the request.

Another way to increase efficiency is through path diversification. This is best described with an example. Let k and j be two nodes that are relatively close to a reference node and must each be resynchronized at the same rate. Assume that when other nodes wish to resynchronize with the reference node, they tend to favor a forwarding path that includes k, but not j. Thus, k itself never needs to send a resynchronization request because it is frequently resynchronized by other requests. On the other hand, j rarely "sees" new synchronization information by virtue of being in the path of other requests and must occasionally generate its own resynchronization request. Path diversification allows requests to be sent through both k and j in proportion to how frequently they require resynchronization. Path diversification can be implemented as follows. Each node knows when it next requires resynchronization. When a node x is forwarding or generating a synchronization request, it forwards the synchronization request to a participating neighbor with the earliest resynchronization deadline.

7. SIMULATION AND RESULTS

7.1 Simulation setup

The simulations were conducted using Omnet++ [14], a discrete event simulator developed by Andras Varga at the Technical University of Budapest. Omnet++ provides built-in support for modeling wireless channels and a notion of time for distributed scheduling of events at nodes. Custom functionality was implemented using C++.

7.2 Implementation Details

Results are based on simulations of connected ad-hoc networks consisting of 500 nodes. The nodes are placed uniformly at random within a 2-dimensional 120m*120m rectangular area. The radio range is is 10m. Additionally, the network contains a single reference node that keeps accurate time. This reference node is placed in the center of the rectangular area. All nodes in the network are aware of their own locations, the location of the refer-

ence node and the locations of their single-hop neighbors. Location information is used only to construct the spanning tree for multi-hop synchronization and to route synchronization requests toward the controller. In the simulation, the depth-first search algorithm is employed to construct the spanning tree.

The simulation was executed for 36,000s or 10 hours. A multi-channel MAC model as described in [15] is assumed. In a multi-channel MAC the frequency is divided into several bands and each node uses a locally unique band to communicate with its neighbors. This MAC model minimizes collisions and thus collisions were considered negligible in the simulation. Further, a very simple channel model is employed. At each attempted packet transmission, independent of all other communication attempts, the success probability for a packet transmission to a neighboring node that is within radio range is Bernoulli with parameter p. In the simulation p is either 0.95 or 0.65.

The sender and receiver delay is modeled as a Gaussian variable with a mean of 0.0001 seconds of and a standard deviation of 11 microseconds. The required accuracy is 0.5 seconds. The drift of the clocks is 50ppm, which is typical for quartz crystals. The required accuracy combined with the drift and synchronization accuracy results in an average inter-synchronization time of about 1000 seconds.

7.3 Results

First we will investigate the efficiency of the synchronization algorithms in terms of the number of pair-wise synchronizations required to keep the network synchronized. Figure 4 shows the total number of pair-wise synchronizations for all nodes in the network using different algorithms. When all nodes are participating the centralized solution has the fewest number of synchronizations. The 18000 synchronizations that occur in the centralized algorithm translate, on average, to 36 synchronizations for each node over the course of 10 hours.

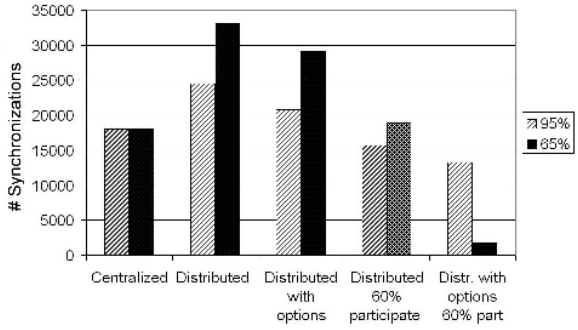

Figure 4: Number of synchronizations for different algorithms and channel quality.

Depending on the channel quality the distributed algorithm adds about 40%-100% overhead compared to the centralized algorithm. Adding options (algorithm enhancements described in Section 6.2) reduces the overhead of the distributed algorithm to 15%-60%. The channel quality (packet transmission success rate) affects the number of synchronizations performed by distributed algorithms greatly because the synchronization requests take longer paths. This leads to an increase the depth of the synchronization trees. As the depths of the synchronization trees increase more frequent synchronization is required. The increase in both the average and maximum depths of the synchronization trees can be seen in Figure 5 and Figure 6.

When only 60% of the nodes are participating the distributed algorithm with enhancements significantly outperforms the other solutions, especially if the channel is good. The use of the distributed algorithm is justified if only a portion of nodes is participating in synchronization.

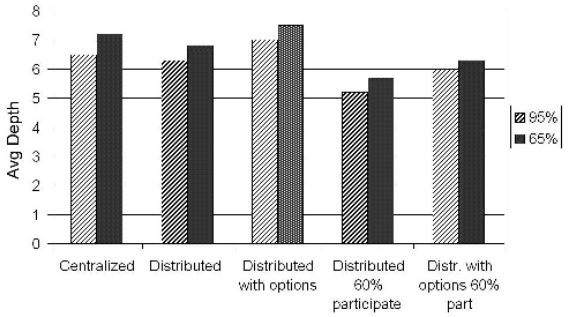

Figure 5: Average depth of synchronization tree for different algorithms

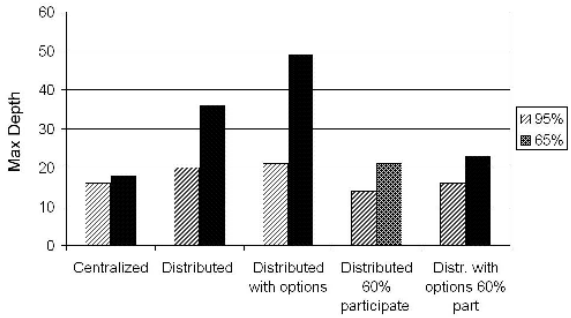

Figure 6: Maximum depth of synchronization tree for different synchronization algorithms.

In addition to the number of synchronizations for the whole network, the number of synchronizations as a function of depth and degree were also explored. Figure 7 shows the percentage of synchronizations performed at various depths in the synchronization tree. Using the centralized algorithm, all nodes perform the same number of synchronizations. Therefore, the centralized curve displays the proportion of nodes at different depths. The percentage of synchronizations performed by the distributed algorithms show a very similar pattern to that of the centralized case. Thus we deduce that the percentage of synchronizations performed at a certain depth is proportional to the number of nodes at that depth and does not show any other strong dependencies on depth.

Further, the relationship between node degree (number of neighbors) and number of synchronizations was explored. Figure 8 presents the number of synchronizations as a function of the degree of the node. For the centralized case the number of synchronizations does not depend on the degree because the synchronization takes place on a spanning tree and each node is synchronized only once, regardless of degree. However, with the distributed algorithm the number of synchronizations tends to increase as a node's degree increases. The node is more likely to receive additional synchronization requests if it has a higher degree. When the degree nears twenty there is a decrease in the number of synchronizations. A possible explanation is that the high-degree

nodes are closely interconnected and when one neighbor starts a synchronization all the other neighbors are aware of this and fall requests are forwarded to the requesting neighbor. The node and most of its neighbors may also be close to the reference node and therefore the node will not receive synchronization requests from its neighbors.

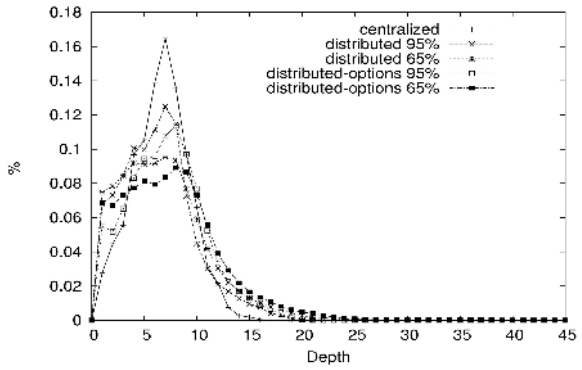

Figure 7: Percentage of synchronizations as a function of depth in synchronization tree

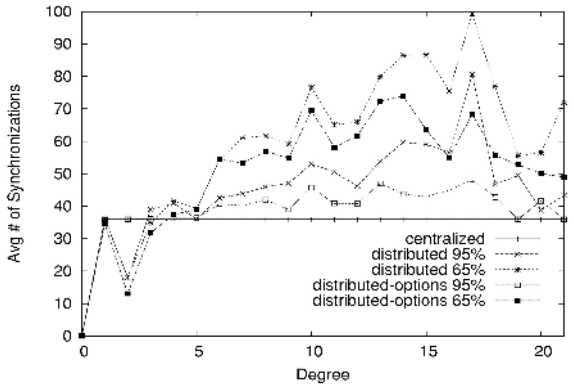

Figure 8: Average number of synchronizations as a function of node degree

The next four graphs relate to the accuracy achieved by the synchronization algorithm. Figure 9 shows the average accuracy, the nodes' offset from the reference time directly after synchronization, as a function of the depth at which the synchronization occurred. The expected linear increase is evident in the figure, confirming that the accuracy of synchronization deteriorates linearly as the hop-distance from the reference node increases.

In addition to post-synchronization accuracy, the offset or error before synchronization is an important parameter. The offset before synchronization gives a bound on the maximum inaccuracy of the nodes. The average offset before synchronization is shown in Figure 10. The average offset peaks at about 0.4 seconds, which is within the accuracy bound of 0.5 seconds. In the distributed case (especially when the enhancement options are used) the average offset for nodes close to the controller is smaller. In the distributed case the nodes near the controller synchronize more frequently because they can have many separate synchronization requests from more distant nodes.

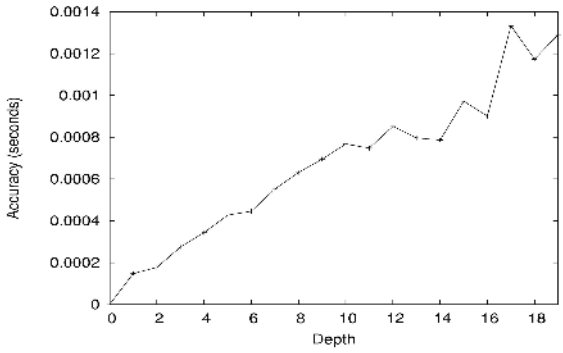

Figure 9: Accuracy of synchronization as a function of node depth in the tree

Figure 11 shows the maximum offset as a function of depth. As the depth increases the maximum tends to be higher and goes above the bound of 0.5 seconds. There is a non-zero probability of this occurring because the bound is statistical.

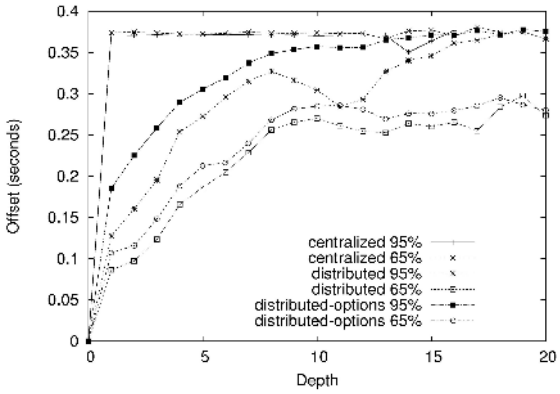

Figure 10:Average time offset before synchronization as a function of node depth in tree

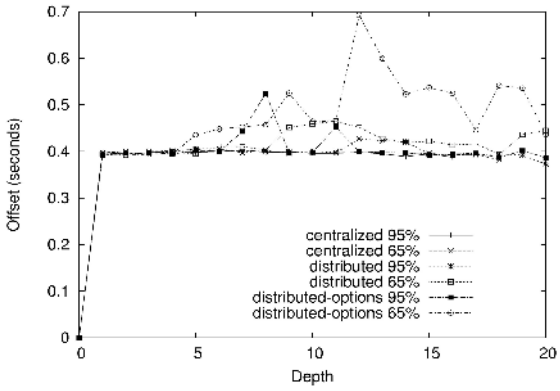

Figure 11:Maximum time offset before synchronization as a function of node depth in tree

8. FUTURE WORK

The LTS schemes presented in this paper rely on the reliability and correctness of information from all nodes along the path to the reference node. Thus, the synchronization will fail if there are Byzantine faults, e.g. clock failure or malicious misinformation

from a subset of nodes. Our algorithm may be updated to function correctly in the presence of these malicious faults.

9. CONCLUSION

In this paper we argue that the required time accuracy of most sensor network applications is relatively low. For applications with lower requirements, the lightweight synchronization scheme we developed is an effective way to give up accuracy for gains in energy efficiency. We show that an algorithm based on pair-wise synchronizations can be used if the desired accuracy is much lower than the accuracy achieved along the longest linear chain of pair-wise synchronization. We investigate the performance of a centralized and distributed LTS scheme. Results indicate that when all nodes participate the centralized scheme is more efficient that the distributed scheme but when a portion of the nodes need frequent synchronization the distributed scheme can result in fewer pair-wise synchronizations. We also show that the scheme is robust and works well in the presence of dynamic links and fading.

10. REFERENCES

[1] J. Rabaey, J. Ammer, T. Karalar, S. Li, B. Otis, M. Sheets, T. Tuan, PicoRadios for Wireless Sensor Networks: The Next Challenge in Ultra-Low-Power Design in *Proceedings of the International Solid-State Circuits Conference*, San Francisco, CA, 2002.

[2] B. Hofmann-Wellenhof, H. Lichtenegger, and J. Collins GPS Theory and Practice, *SpringerWienNewYork,* 1997.

[3] D. Mills, Network Time Protocol (Version 3) Specification, Implementation and Analysis, from *http://www.faqs.org/ftp/rfc/rfc1305.pdf.*

[4] E. Anceaume and I. Puaut , A Taxonomy of Clock Synchronization Algorithms, *Research report IRISA, NoPI1103*, July 1997.

[5] 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,* Boston, MA. December 2002.

[6] M.L. Sichitiu and C. Veerarittiphan, Simple, Accurate Time Synchronization for Wireless Sensor Networks. *IEEE Wireless Communications and Networking Conference, WCNC* 2003

[7] Saurabh Ganeriwal, Ram Kumar, Sachin Adlakha and Mani Srivastava, "Network-wide Time Synchronization in Sensor Networks," *Technical Report* UCLA, April 2002.

[8] S. Mitra and J. Rabek, Power Efficient Clustering for Clock Synchronizarion in Dynamic Multi-hop Sensor Networks, from *http://theory.lcs.mit.edu/~mitras/courses/6829/project/project_main.html.*

[9] J. Elson and K. Römer, 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.

[10] H. Kopetz, W. Schwabl. Global time in distributed real-time systems. *Technical Report 15/89,* Technishe Univesität Wien, 1989.

[11] Warneke, B. Atwood, K.S.J. Pister, Smart Dust Mote Forerunners, *Proceedings of the Fourteenth Annual International Conference on Microelectromechanical Systems (MEMS 2001),* Interlaken, Switzerland, January 21-25, 2001, pp. 357-360.

[12] B. Awerbuch, A new distributed depth first search algorithm, *Inf. Proc. Lett. 20* (1985), 147-150.

[13] A. Boukerche, C. Tropper, A Distributed Graph Algorithm for the Detection of Local Cycles and Knots, *IEEE Trans. Parallel and Distributed Systems*, 1998, pp. 748-758

[14] A. Varga, "The OMNeT++ Discrete Event Simulation System," in *European Simulation Multiconference* (ESM'2001), Prague, Czech Republic, June 2001.

[15] 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