MTSF: A Timing Synchronization Protocol to Support Synchronous Operations in Multihop Wireless Networks

Jungmin So
Dept. of Computer Science, and
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
Email: jso1@uiuc.edu

Nitin Vaidya
Dept. of Electrical and Computer Engineering, and
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
Email: nhv@uiuc.edu

Technical Report

October 2004

***Abstract*— In this paper, we propose a protocol for synchronizing time in multihop wireless networks. Protocols for power management or dynamic channel assignment often require synchronous operations, such as waking up at the same time or listening to a common channel. Having the nodes synchronized in time is often crucial for these protocols to work. However, the existing synchronization mechanisms do not work well with these protocols because either they may fail to synchronize the time even without transient failures or packet loss, or they may require a large overhead. The proposed synchronization protocol, called MTSF (Multihop Timing Synchronization Function), successfully synchronizes time in a multihop network, at a low cost. In the absence of packet loss, MTSF guarantees an upper bound on the clock error between any two pair of nodes in the network. Moreover, MTSF can tolerate packet losses to some extent and its performance degrades gracefully when the loss rate becomes high. The cost of MTSF increases very slowly as the number of nodes increase, which makes the protocol highly scalable. MTSF is fully distributed, and operates without a central coordinator. Finally, MTSF is self-stabilizing, which means that starting from an arbitrary state, the protocol converges to a steady state. Due to these features, MTSF can support protocols that require synchronous operations at a low cost.**

I. Introduction

Time synchronization is an important feature in distributed systems. Many applications and protocols require synchronous operations, and those operations rely on synchronized time. One example is the power saving mechanism (PSM) of IEEE 802.11 [1]. In IEEE 802.11 PSM, all nodes wake up at the beginning of a beacon interval to exchange messages. While the nodes are awake, they exchange messages to schedule transmission for the current beacon period. If a node does not have any communication scheduled, it may turn off its radio and go to sleep for the rest of the beacon period. When a node wakes up at the start of a beacon period, it expects other nodes to be up as well. However, if the nodes are not time-synchronized, nodes may wake up at different times, causing packet losses when they start transmitting packets.

To synchronize time, IEEE 802.11 has a protocol called Timing Synchronization Function (TSF), which is explained in detail in Section IV. TSF uses timestamped beacon messages transmitted at the start of each beacon period to synchronize clocks among nodes. The protocol is designed for wireless LANs, where there is a direct link between any pair of nodes. Huang and Lai [2] points out that TSF is not scalable, meaning that as the number of nodes increase, the possibility that the nodes go out of synchronization becomes large enough to significantly impact protocols that rely on synchronized clocks. They propose a simple modification to TSF to improve the scalability, as explained in Section III.

For a multihop network, it is much more difficult to achieve time synchronization, because nodes are spread out in multiple broadcast domains. IEEE 802.11 TSF does not work in multihop network, mainly due to the reason that beacon messages sent on different broadcast domains do not agree with each other. As elaborated later, this leads to a problem we call *time partitioning* where the time in two groups of nodes can keep on drifting away from each other, even though they are connected.

Several clock synchronization protocols have been proposed for multihop wireless networks, including sensor networks. They are summarized in Section III. However, as discussed in Section III, these protocols do not support the synchronous operations well. Some protocols have low overhead but cannot avoid time partitioning problem, and other protocols that achieve global accuracy incur too much overhead.

Due to the difficulty of clock synchronization in a multihop network, protocol designers often avoid synchronous operations [3], or assume that the clocks are synchronized using out-of-band mechanisms such as GPS [4]. Also, some protocols introduce additional overhead to avoid relying on time synchronization. One example in this approach is STEM [5], which uses a second channel.

To support synchronous operations, we need a protocol that

This research was supported in part by Motorola.
A revised version from a technical report in January 2004.

maintains the clock error under a certain bound in a stable manner so that this bound can be used for other protocols running on top of this synchronization protocol. Thus, the protocol we propose in this paper, aims to achieve *stable synchronization* at a low cost. By stable synchronization, we mean that the maximum clock error between any pair of nodes in the network does not go over a certain bound for a long time. As shown from the simulation results, MTSF achieves stable synchronization among nodes, with a cost that increases very slowly as the number of nodes increase in a given area.

The rest of the paper is organized as follows. In Section II, we describe the problem formally and present the goal we try to achieve in this paper. In Section III, we review the existing clock synchronization protocols. In Section IV, we describe IEEE 802.11 TSF in detail and identify its problems to provide insights that lead to our proposed protocol. In Section V, we describe our proposed protocol, MTSF, and provide mathematical proofs of its features. In Section VI, we evaluate the performance of MTSF using simulations. Finally, we draw our conclusion in Section VII.

II. Problem Definition

In this section, we describe formally the problem addressed in this paper. We also define terms and variables that will be used throughout the paper.

We consider an ad hoc network that consists of multiple wireless nodes. The network may span multiple hops, meaning that a pair of nodes may be connected via other nodes acting as intermediate relays. No infrastructure exists. We assume that the network is always connected, meaning that when all nodes are active (not sleeping), one can find a path between any pair of nodes in the network.

Each node maintains a hardware clock. The value of the hardware clock is called *physical time* and the physical time of node i is denoted as T_i^P. We also assume that there exists a "real" clock, which represents the real time. The nodes have no knowledge on the real time. We denote the real time as t, and we can express the relationship between physical time of node i and real time as the following.

$$T_i^P = \alpha_i t + \beta_i \tag{1}$$

In Equation 1, both α_i and β_i are both determined by hardware clock and cannot be controlled by the protocol.

Other than the hardware clock, each node also maintains a software clock. The value of the software clock is called *logical time*. Logical time can be modified by the protocol. The following equation describes the relationship between physical time and logical time. The logical time of node i is denoted as T_i.

$$T_i = T_i^P + \gamma_i \tag{2}$$

In the equation, γ_i is the parameter that can be controlled by the synchronization protocol. A node can correct the logical time to reduce the time difference with other nodes, and this process is called *time synchronization*. So when we say node A synchronizes to node B, it means that node A adjusts the γ value to reduce $|T_A - T_B|$. Using the above two equations, we can state the relationship between logical time and the real time as follows.

$$T_i = \alpha_i t + \delta_i \tag{3}$$

where $\delta_i = \beta_i + \gamma_i$. In the above equation, α is called the *clock rate*, and δ is called the *clock offset*. From here on, when we refer to the "time" at a node, we mean logical time at that node.

Suppose all nodes start their clock exactly at the same time, with the same initial time. Since each node has a different clock rate α, after some time all nodes will have different logical time. In order to synchronize the time, each node exchanges messages telling their local time to other nodes. Then nodes use this messages to adjust their clocks so that the time difference among nodes is kept small[1]. Due to the uncertainty in message delay, the nodes in the network cannot be synchronized to the exact same time. We define *clock error* to be the time difference between a pair of nodes, and we denote the clock error of node i and j as Δ_{ij}. Specifically,

$$|T_i - T_j| = \Delta_{ij} \tag{4}$$

Also, we define the maximum of all the clock errors as the *global clock error*, which is denoted as $\max \Delta_{ij}$. The goal of a synchronization protocol is to make the global clock error small so that it can be always kept under a certain threshold, which we call the *synchronization threshold*. The synchronization threshold represents the amount of accuracy an application requires. Specifically, we define our goal of time synchronization as follows. For a given synchronization threshold W,

$$\max \Delta_{ij} \leq W \tag{5}$$

If the protocol guarantees this upper bound on the global clock error, an application that runs on top of this protocol can use this information to set up a "margin" before starting a synchronized operation. For example, in a power management scheme, assume that the time is divided into beacon intervals, and node A and B are supposed to wake up at the beginning of each beacon interval and exchange messages. If A knows that $\Delta_{ij} \leq W$, then A can wait for W before transmitting a packet, after A starts a new beacon interval. This is depicted in Figure 1.

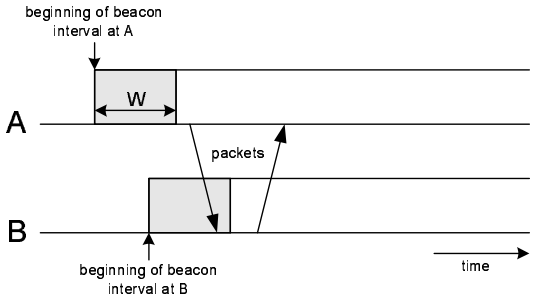


Fig. 1. Node A knows that the maximum synchronization error is W, so node A waits for duration W at the beginning of its beacon interval before transmitting.

[1] time = logical time

III. Related Work

Numerous time synchronization schemes exist in the context of wired and wireless networks [6], [7], [8], [9], [10], [11], [12], [13]. In this section, we review representative works in this area.

In wired networks, clock synchronization is often achieved using the Network Time Protocol (NTP) [6]. In NTP, multiple canonical sources are used as reference clocks, and a hierarchical structure is built that are rooted at these sources. Timestamped packets are exchanged along the branches of the trees, so that all nodes can synchronize to one of the canonical sources.

NTP is designed for wired networks, with the assumption that the network is mostly static. So NTP pre-configured synchronization hierarchy. However, in a wireless networks, topology may change frequently due to mobility, node failures and link failures. Also, predefined canonical sources may not exist in an ad hoc network. Thus NTP cannot be directly applied to wireless ad hoc networks.

For wireless LANs, IEEE 802.11 standard has a synchronization protocol called timing synchronization function (TSF) [1]. TSF is used for power saving mode (PSM) where every node has to wake up at the same time (beginning of a beacon interval) to exchange messages. At the beginning of a beacon interval, each node picks a random delay before transmitting a beacon. When a node transmits a beacon, all other nodes receives the beacon, suppress their beacon transmissions, and synchronize to the beacon sender using the timestamp included in the beacon. A node only synchronizes to a faster node (a node with a faster logical time) to avoid going back in time. IEEE 802.11 TSF is described in more detail in the next section, as it is relevant to our proposed protocol.

This scheme has the problem of *fastest node asynchronism*, identified by Huang and Lai [2]. Since a node only synchronizes to a faster node, the time of the fastest node (a node with the fastest logical time in the network) will keep drifting away from other nodes, unless it becomes the beacon transmitter. As the number of nodes increases, the chance that the fastest node transmits becomes smaller. Huang and Lai propose a simple modification to TSF called ATSP (Adaptive Timing Synchronization Procedure), to reduce the impact of fastest node asynchronism [2]. The idea is to have each node adjust their frequency of beacon transmission according to the received beacon messages. In each beacon interval, if a node receives a beacon with a faster clock, it reduces its beacon frequency. If not, then it increase its beacon frequency until it reaches the maximum. This scheme works well in wireless LAN. However, when these schemes are applied to multihop networks, the clocks might still drift away because of the *time partitioning* problem as explained in the next section.

For multihop networks, Sheu et al. [14] proposed a scheme called Automatic Self-time-correcting Procedure (ASP). This protocol has two features. First, as in ATSP, the frequency of beacon transmission is adjusted according to the relative time values among neighbors, so that a faster node transmits beacon with higher probability than a slower node. Second, a node estimates the clock rate difference with its neighbor who is a faster node, so that it can automatically adjust its clock even when it does not receive a beacon from a faster node. This protocol improves the synchronization accuracy without incurring additional overhead compared to IEEE 802.11 TSF. However, it is not easy to obtain an accurate estimation on the neighbor's clock rate. First, the clock rate can change due to environmental changes such as temperature. Second, this protocol does not consider estimation error due to propagation delay, but even small estimation error can lead to significant error in estimating the clock rate.

Ganeriwal et al. proposed a scheme similar to NTP, but modified to work in sensor networks [12]. The protocol, called Timing-Sync Protocol for Sensor Networks (TPSN), builds a tree structure in the network so that all nodes synchronizes to the root node. To synchronize a pair of nodes, TPSN uses pairwise message exchange to reduce estimation error. This protocol achieves good accuracy, but each synchronization round requires $2n$ transmissions, when the number of nodes is n. This is a high overhead when the network is dense. Also, TPSN does not deal with fastest node asynchronism. It is often the case that clocks are not allowed to go back in time, because the local ordering of events cannot be preserved. The lightweight tree-based synchronization algorithm (LTS) [11] is similar to TPSN, but each node chooses the synchronization period based on the desired accuracy.

Until now, all protocols assumed that a node can stamp the time at the MAC layer, just before transmitting the packet. So the uncertainty in delay for contending the channel is removed. The Reference-Broadcast Synchronization (RBS) [8], considers this delay, because it assumes that the timestamping is done at a higher layer. To remove the uncertainty in the delay before a node gains channel access, the protocol proposes receiver-receiver synchronization paradigm. The idea is to have a reference node broadcast packet, and receivers compare their observations to synchronize time to each other. This protocol achieves high accuracy, but at a cost of high overhead. Also, multihop synchronization is not well-defined, although they propose to use nodes in an overlapping area of two broadcast domains to transfer time information.

Li and Rus proposes three mechanisms for achieving clock synchronization in sensor networks [13]. The all-node based synchronization and cluster-based synchronization are not scalable nor fault-tolerant as argued in [13]. In the diffusion-based algorithm, each node periodically reads its neighbors' clocks and computes average. Then it returns the average value to its neighbors. This scheme has a tradeoff between convergence time and overhead.

Finally, Elson and Estrin proposed the concept of *post-facto synchronization* [7]. In post-facto synchronization, the clocks are left unsynchronized. When an event happens, the relevant nodes coordinate with each other to figure out what event happened at what time. On the other hand, in a priori synchronization, nodes exchange messages to maintain clocks synchronized. Post-facto synchronization can preserve ordering of events, but cannot be used to support synchronous operations, because the clocks need to be synchronized prior to the operation.

Our protocol extends IEEE 802.11 TSF to work in multihop networks by having each node maintain a soft state. Since TSF

is highly relevant to our protocol, we examine the protocol in a greater detail in the next section.

IV. IEEE 802.11 Timing Synchronization Function (TSF)

In this section, we describe IEEE 802.11 TSF in detail, and discuss issues when this protocol is applied to a multihop network.

In IEEE 802.11 TSF, the synchronization takes place in every beacon period. At the beginning of a beacon period, each node waits for a random delay before transmitting a beacon. When a node transmits a beacon, the nodes that receives the beacon suppress their beacon transmissions. So for a wireless LAN, only a single packet is transmitted in each beacon period.

Before transmitting a beacon, the sender records the timestamp in the beacon packet using its clock. The timestamp is generated just before the node transmits the packet, so that the uncertainty in the MAC contention delay can be removed. When a node receives the beacon packet, it reads the timestamp and estimates the current time of the sender's clock considering the transmission and propagation delay. If the receiver has a slower time, it synchronizes to the sender by adjusting its time. Node do not synchronize to a slower node to avoid going back in time.

Suppose node A transmits a beacon and B receives it. If A's clock is faster than B's, B adjusts its time to match that of A. Since there is a delay between the point of time node A stamps its time in the beacon packet, and the time B receives the beacon. This delay consists of the transmission time and the propagation delay, as illustrated in Figure 2.

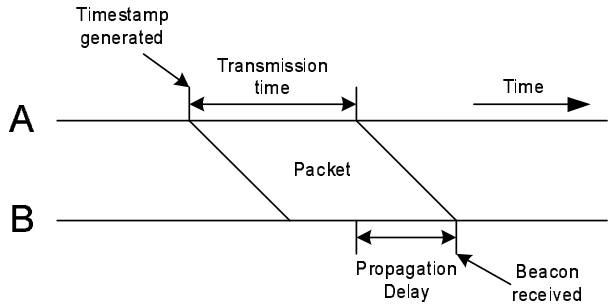


Fig. 2. When node B receives a beacon from node A, it has to estimate A's current time considering the transmission time and the propagation delay.

The transmission time can be measured using transmission rate and packet size. Let B be the transmission rate and p be the packet size. Then the transmission time can be calculated as

$$T_t = p/B \tag{6}$$

Since we know the transmission rate and the packet size, we can obtain the accurate estimation of this delay. However, due to the difference between rate of the receiver's clock and rate of the real clock, an estimation error occurs. The error in estimated transmission time, ϵ_t, is

$$\epsilon_t = |\alpha_R - 1| \times T_t \tag{7}$$

The IEEE 802.11 standard requires the clock accuracy to be within $\pm 0.01\%$. So we assume that the clock rates are within the range [0.9999, 1.0001]. Also, we assume that the size of a beacon packet is 56 bytes, which consists of 24 bytes of preamble, and other 32 bytes of data. Finally, we assume that the preamble is transmitted at 1Mbps, and data is transmitted at 2Mbps. Then, in Equation 7,

$$\max |\alpha - 1| = 0.0001 \tag{8}$$

and,

$$T_t = \frac{192}{10^6} + \frac{256}{2 \times 10^6} = 320\mu s \tag{9}$$

So the maximum estimation error for the transmission time is

$$\max \epsilon_t = 320 \times 0.0001 = 0.032\mu s \tag{10}$$

To estimate the propagation delay, we need to know the distance from the source to the destination. Since the distance is unknown, we use the upper bound as an estimate. Then the maximum estimation error for the propagation delay is

$$\max \epsilon_p = d_{max} \times \frac{1}{C} \tag{11}$$

where d_{max} is the maximum transmission range and C is the speed of light. If we assume d_{max} to be 250m, then the maximum error for propagation time would be approximately 0.8 μs.

On the whole, the maximum estimation delay, ϵ_{max} is

$$\epsilon_{max} = \max \epsilon_t + \max \epsilon_p \tag{12}$$

With 2Mbps of channel bandwidth, 56 bytes of packet size and the transmission range of 250m, the maximum error in synchronizing a pair of nodes is approximately 1 mus.

The IEEE 802.11 TSF is efficient in terms of communication cost, because only one packet is transmitted for each broadcast domain in each beacon period. However, when applied to multihop networks, TSF may fail to synchronize the clocks due to the *time partitioning* problem. Consider the scenario in Figure 3. Suppose node A is faster in time than B, and node D is faster than C. Then node A and D have higher chance of transmitting beacons before node B and C. So if A and D transmit beacons, B synchronizes with A and C synchronizes with D. If this happens for several period, the time between (A,B) and (C,D) will drift away unboundedly. We call this problem *time partitioning*, because even though these two groups of nodes are connected with each other, they do not exchange time information. If node A has a higher clock rate than D, for these two groups to synchronize to each other, B has to transmit a beacon, so that it can propagate to node C and then D. When the number of nodes increases, the problem of time partitioning has an significant impact on the clock accuracy, as shown by simulations in Section VI. Also, note that giving faster nodes higher chance in beacon transmission such as in ATSP increases the impact of the time partitioning problem.

To prevent the time clustering problem and maintain the clocks synchronized, we need to make sure that every node is

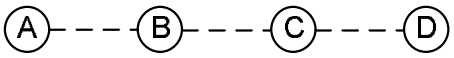


Fig. 3. A simple chain topology with 4 nodes.

synchronized with the fastest node within a certain period. Suppose in Figure 3, node A is the fastest node. Then for node D to synchronize with node A, the time information has to propagate through node B and C. So if we preserve the rule that only one node transmits in a broadcast region, node A has to transmit in the first beacon interval, node B in the next interval, and finally node C in the next interval for D to synchronize with A. This is similar to establishing a path between nodes, so that the packet is forwarded one hop at each beacon interval.

By maintaining a small amount of soft state at each node, we can establish an implicit path from the fastest node to all other nodes. This is what our proposed protocol does, and it is explained in detail in the next section.

V. Multihop Timing Synchronization Function (MTSF)

The basic idea of MTSF is to have each node maintain a path to the fastest node in the network, and make the time of the fastest node propagated through the path, so that every node can synchronize with the fastest node within a certain period of time. For example, in Figure 3, if node A is the fastest node, we want to achieve a schedule as in Figure 4. As we can see in the figure, node D can update its time in every other beacon interval to match the rate of node A. We achieve this schedule by having each node maintain a soft state.

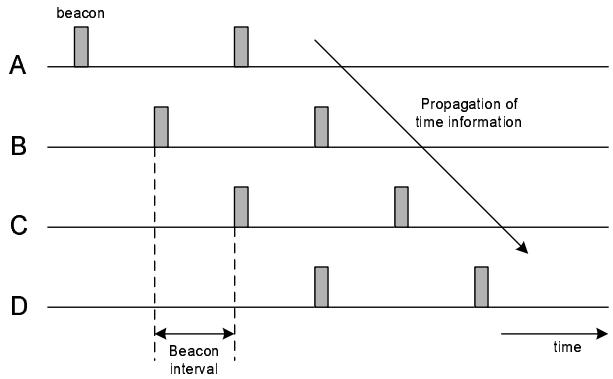


Fig. 4. An example beacon transmission schedule.

In MTSF, each node schedules beacon transmission every two beacon intervals, or *rounds*. At the beginning of a round in which the node schedules beacon transmission, it waits for a random delay before transmitting a beacon. As in IEEE 802.11 TSF, the sender stamps its time in the packet just before transmitting the packet. When a node receives the beacon, it compares the clock of the sender and itself considering the transmission time and propagation delay. If the sender's clock is faster, the receiver synchronizes to the sender. After receiving a beacon, a node decides if it should suppress its own beacon according to the rule explained later.

Every node maintains a "parent" variable, initially set to the identifier of itself. When a node finds a neighbor node with a faster clock, it sets the "parent" variable to be the identifier of the faster node. If there are multiple faster nodes in the neighborhood, the fastest node among those neighbors are chosen as the parent node.

Once the parent node is chosen, the node schedules its beacons in the rounds that its parent node does not schedule beacons. So if the parent node schedules beacons on "odd-numbered" rounds, the child node schedules beacons on "even-numbered" rounds.

Using this simple scheme, each node eventually establishes a path towards the fastest node. When every node in the network has a path between itself and the fastest node, we say that the protocol as converged to a *steady state*. We will prove that starting from an arbitrary state where each node has an arbitrary time, this protocol converges to the steady state, given that the network topology does not change during the process of convergence. Before proving this self-stabilization property, we first derive the upper bound on the clock error between a node and the fastest node in the network given that the protocol is in a steady state. The result is used to prove the self-stabilization property of MTSF.

Lemma 1: When the protocol reaches a steady state, the upper bound on the global clock error is $2f(D+1)L + D\epsilon_{max}$, where f is maximum rate difference compared to the real clock, D is the network diameter, L is the length of a beacon interval, and ϵ_{max} is the maximum estimation error.

Proof: Suppose the fastest node in the network is R, and the node we want to calculate the clock difference is k hops away from R, in the path established by the protocol. Note that k may not necessarily be the shortest hop distance between these two nodes. We name the node C_k, and the nodes in the path from R to C_k are C_1, C_2, ..., C_{k-1}.

To make the equations simple, we ignore the impact of estimation error ϵ_{max} (defined in Section IV in the analysis. The effect of ϵ_{max} is added to the upper bound at the end of the analysis.

At ith beacon interval, node R sends a beacon to node C_1, and C_1 synchronizes to node R. After the synchronization, $T^i_{C_1} = T^i_R$. Then at the beginning of $(i+1)$th interval, T_{C_1} can be expressed as follows.

$$T^{i+1}_{C_1} = T^i_R + \alpha_{C_1} L \tag{13}$$

where α_{C_1} is the clock rate of C_1 and L is the length of the beacon interval. Now at $(i+1)$th interval, C_1 sends a beacon and C_2 is synchronized to C_1. Since C_1 is the fastest neighbor of C_2, C_1 is the last node that C_2 synchronizes to in the beacon interval. Then at the beginning of $(i+2)$th interval, the time of C_2 is

$$T^{i+2}_{C_2} = T^{i+1}_{C_1} + \alpha_{C_2} L = T^i_R + (\alpha_{C_1} + \alpha_{C_2})L \tag{14}$$

Continuing this process, node C_{k-1} will send a beacon at $(i+k-1)$th beacon interval, and C_k will synchronize to C_{k-1}. Then at the beginning of $(i+k)$th beacon interval, the time of C_k is

$$T^{i+k}_{C_k} = T^i_R + (\sum \alpha_{C_j})L \tag{15}$$

where $j = 1, 2, \ldots, k$. Also, since node R never synchronizes to other nodes,

$$T_R^{i+k} = T_R^i + \alpha_R k L \tag{16}$$

Thus, the clock difference between C_k and R at the beginning of $(i+k)$th beacon interval is

$$\Delta_{C_k R} = T_R^{i+k} - T_{C_k}^{i+k} = \alpha_R k L - (\sum \alpha_{C_j}) L = (\sum (\alpha_R - \alpha_{C_j})) L \tag{17}$$

where $j = 1, 2, \ldots, k$.

Since a node synchronizes to its parent node in every other beacon interval, C_k does not synchronize to C_{k-1} in $(i+k)$th beacon interval. So at the beginning of $(i+k+1)$th beacon interval, the clock difference between C_k and R becomes the maximum, and it is

$$\Delta_{C_k R} = (\sum (\alpha_R - \alpha_{C_j}) + (\alpha_R - \alpha_{C_k})) L \tag{18}$$

This is the maximum clock error between a node and the fastest node in the network.

Suppose D is the network diameter, which is the maximum hop distance between any pair of nodes in the network. Then the maximum clock error among all pairs of nodes in the network will be

$$\max \Delta = (\sum (\alpha_R - \alpha_{C_j}) + (\alpha_R - \alpha_{C_D})) L \tag{19}$$

where j = 1, 2, ..., D, and C_1, C_2, ..., C_{D-1} are the nodes in the path from R to C_D.

Since the clock rates are unknown, a node cannot precisely determine $\max \Delta$. So we can consider the worst case, where R has the maximum clock rate and all other nodes have the minimum allowable clock rate. If the clock rate is required to be within the range $[1-f, 1+f]$, then $\max \Delta$ becomes

$$\max \Delta = (\sum (\alpha_R - \alpha_{C_j})) L = 2f(D+1)L \tag{20}$$

Now we take into account ϵ_{max}. Since the maximum estimation error for each hop is ϵ_{max}, the new upper bound on the network synchronization error is

$$\max \Delta = 2f(D+1)L + D\epsilon_{max} \tag{21}$$

■

If we assume f is 0.0001, D is 10, L is 100ms, and ϵ is 1μs, then the maximum network synchronization error will be 230 μs.

This is a very conservative calculation of the global clock error because we assumed that the fastest node has the maximum clock rate, and all other nodes have the minimum clock rate. Thus, in reality, MTSF may achieve a lower bound on the accuracy in the steady state.

Now we prove that the protocol converges to a steady state, where every node in the network establishes a path to the fastest node.

Lemma 1: Starting from an arbitrary state, the protocol eventually enters a steady state, where every node in the network establishes a path to the fastest node. In the steady state, every node updates its time in every two beacon intervals to match the rate of the fastest node.

Proof: If node S has node D as its parent, we say that node S "points to" node D. Also, if node S can reach node R by going up the path, we say that node S "points towards" node R.

We prove that if a node always chooses the fastest neighbor as its parent, it eventually points towards the fastest node in the network. When every node points towards the fastest node, the protocol enters a steady state.

We start with a simple example and generalize the argument to any possible cases.

Consider the scenario in Figure 5. The clock rate of node A, B, C, D and E is α_A, α_B, α_C, α_D, and α_E, respectively. Suppose $\alpha_A > \alpha_E > \alpha_B > \alpha_D > \alpha_C$. At some point of time, B will regard A as the fastest node in the network, because node A has a faster rate than B. On the other hand, D will regard E as the fastest node. After that, node C has to decide whether it should pick node B or node D as its parent. We argue that if C always chooses a faster node as its parent, C will eventually choose B as its parent. Applying the same argument, D and E will also eventually point toward node A.

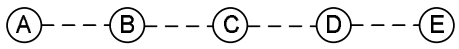


Fig. 5. A network scenario with 5 nodes placed in a chain topology.

To show this, we consider the clock difference Δ_{BC} and Δ_{CD} at the start of kth beacon interval. Considering all possible situations, we show that eventually Δ_{BC} becomes larger than Δ_{CD} so that node C chooses B as its parent. For simplicity of the analysis, we ignore the impact of the estimation error, ϵ_{max}. However, the argument below still holds even if the estimation error is taken into account.

Let L be the length of a beacon interval. Assume that in $(k-1)$th beacon interval, node A broadcasts a beacon, and B synchronizes to A. So at the start of kth beacon interval,

$$\Delta_{AB} = (\alpha_A - \alpha_B) L \tag{22}$$

Ignore nodes D and E for now. At the kth beacon interval, node B transmits a beacon and node C synchronizes to node B. So at the start of $k+1$th beacon interval,

$$\Delta_{BC} = T_B - T_C = T_A - \Delta_{AB} - T_C = \tag{23}$$

$$\alpha_A t + \delta_A - (\alpha_A - \alpha_B) L - (\alpha_C t + \delta_C) \tag{24}$$

We can rewrite the equation as

$$\Delta_{BC} = (\alpha_A - \alpha_C) t + u \tag{25}$$

where u is a constant ($u = \delta_A - (\alpha_A - \alpha_B) L - \delta_C$). Similarly,

$$\Delta_{CD} = (\alpha_E - \alpha_C) t + v \tag{26}$$

where v is a constant. Since $\alpha_A > \alpha_E$, as t increases, eventually Δ_{BC} becomes larger than Δ_{CD}. Thus, node C eventually chooses node B as its parent.

We can generalize this argument and prove that every node in the network will eventually choose its parent towards the fastest node, if each node chooses the fastest neighbor as its parent.

We prove this using the previous argument and by induction on hop distance of a node from the fastest node. Let R be the fastest node in the network. For the base case, suppose node A_1 is a one-hop neighbor of node R. Since node R is the fastest node, for any node i in A_1's neighbor set.

$$\Delta_{A_1 R} \geq \Delta_{A_1 i} \tag{27}$$

Now suppose node A_{k+1} is $k+1$ hops away from node R. It has a neighbor node A_k, which is already pointing towards node R. All of A_k's ancestors, A_1, A_2, ..., A_{k-1}, are pointing toward node R.

Since the nodes A_i ($i = 1, 2, ..., k$) are already pointing toward node R, from Equation 18, the maximum of $\Delta_{A_k R}$ at the start of a beacon interval is

$$\max \Delta_{A_k R} = (\sum(\alpha_R - \alpha_{A_i}) + (\alpha_R - \alpha_{A_k}))L \tag{28}$$

where $i = 1, 2, ..., k$. Thus,

$$\min \Delta_{A_{k+1} A_k} = T_R - ((\sum(\alpha_R - \alpha_i) + (\alpha_R - \alpha_{A_k}))L) - T_{A_{k+1}} \tag{29}$$

Thus, $\Delta_{A_{k+1} A_k}$ can be written as

$$\Delta_{A_{k+1} A_k} = (\alpha_R - \alpha_{A_{k+1}})t + c \tag{30}$$

where c is a constant. Similarly, if another neighbor of node A_{k+1}, B, is pointing towards node S, then

$$\Delta_{A_{k+1} B} = (\alpha_S - \alpha_{A_{k+1}})t + c' \tag{31}$$

Since α_R is greater than α_S, eventually node A_{k+1} will choose A_k as its parent.

So if every node always chooses the fastest neighbor as its parent, then eventually every node will point towards the fastest node in the network, and the protocol enters a steady state. ■

Until now, we have shown that if each node transmits beacon in every other beacon interval and choose the fastest neighbor as its parent, all nodes will eventually point towards the fastest node in the network. However, if every node transmits beacon in every other beacon interval, the communication overhead of this protocol is proportional to the number of nodes, which is not scalable.

The reason for having each node maintain the "parent" variable is to reduce the communication overhead while still preserving the upper bound on the global clock error. We define *leaf node* to be a node which does not have any child that is pointing towards itself. The leaf nodes do not contribute to the accuracy of the protocol, so they do not need to transmit beacons every other beacon interval.

However, if a leaf node does not transmit beacon at all, the protocol will not be able to adapt to change in the topology. Suppose a new node joins the network and it needs to point to the leaf node to reach the fastest node. If a leaf node does not transmit at all, the new node would not be able to synchronize to the fastest node. Thus, leaf nodes can transmit at a low frequency to advertise its existence, but not in every other beacon interval.

To identify whether a node S is a leaf node or a non-leaf node, we need feedback from the nodes who consider S as their parent. So every node includes its parent identifier in the beacon packet, so that when a node receives a beacon with the parent identifier as itself, then the node knows that it is a non-leaf node.

The beacon transmission rules for non-leaf nodes and leaf nodes are as follows. If a node is a non-leaf node, it transmits beacons in every other beacon interval, regardless of whether it receives a beacon in that interval or not. If a node is a leaf node, it suppresses its beacon if it receives a beacon from *another leaf node with the same parent* in the beacon interval. This is to make sure that their parent node is notified of their existence. In addition, to guarantee that every leaf node transmits a beacon eventually, we force every leaf node to transmit beacon with a probability p even though they receive a beacon from another node sharing the same parent. If a non-leaf node does not receive beacons from any of its children for several consecutive beacon intervals, it regards itself as a leaf node.

This scheme reduces the communication cost of MTSF significantly, while maintaining the upper bound on the clock error. As we will see in the next section, the number of non-leaf nodes grows slowly as the number of nodes increase in a given area. So the communication overhead of MTSF grows slowly with increasing node density, which makes MTSF a scalable protocol.

VI. Performance Evaluation

In this section, we report results from the simulations we performed to study the performance of MTSF. For the simulations, we have used our own simulator written in C++. We want to see if MTSF successfully bounds the clock error between any pair of nodes, and if MTSF achieves this goal at a low cost. For comparison, we also simulate a modified version of IEEE 802.11 TSF. In the modified version of IEEE 802.11, when a node receives a beacon, it does not always suppress its beacon, but transmits the beacon with a probability p. If p is 0, then the protocol falls back to the basic IEEE 802.11 TSF. If p is 1, then every node transmits a beacon in a beacon period.

To measure the performance of a protocol in terms of clock accuracy, we use the following metrics.

- Global clock error: This is the maximum of clock difference between any pair of nodes in the network. We trace the maximum clock error over time to see the behavior of the protocol. Tracing maximum clock error also shows how fast the protocols converge, because at the beginning of each simulation, the clock values are arbitrarily chosen for each node.
- Percentage of time that the network is out of synchronization: For different threshold values, we measure the percentage of time the maximum clock error exceeds the threshold.

To measure the protocol overhead, we use the following metrics.

- Average number of beacon transmissions per round in a broadcast domain: For each round, we measure how many beacons are transmitted in a broadcast region. For example, if a node receives three beacons in a round, the number of beacon transmitted in that broadcast region is 3. It

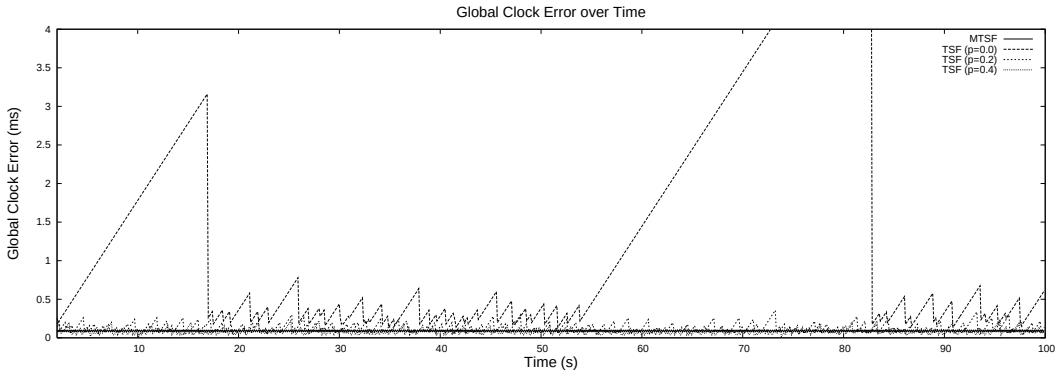


Fig. 6. Global Clock Error over time.

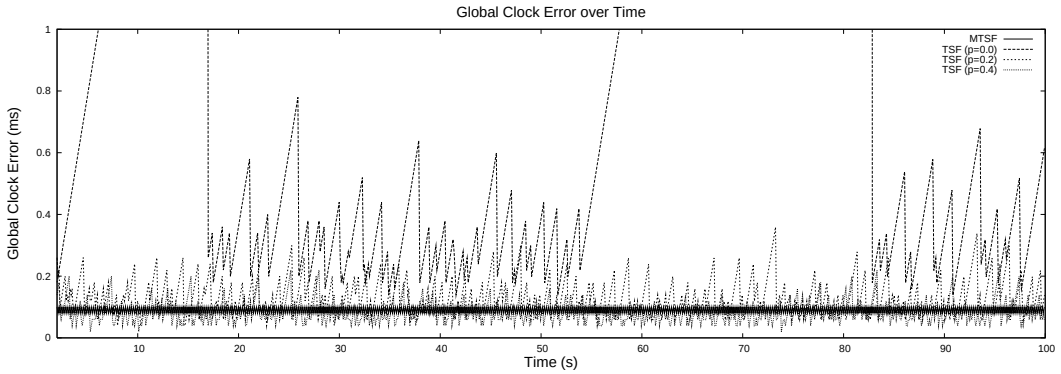


Fig. 7. Global Clock Error over time. Shown with a smaller scale on Y-axis.

is averaged over all nodes and over the whole simulation time.

- Percentage of leaf nodes in the converged synchronization tree: This metrics indicates how MTSF builds an efficient synchronization tree in the network.

We measure these metrics with different network size and packet loss rate to see the impact of these factors. We first describe our simulation setup, and then we present and discuss the results.

A. Simulation Setup

In all of our simulations, nodes are randomly placed in a square-shaped region. The size of the area is 1000m $\times$ 1000m, unless otherwise specified. Every node has a fixed transmission range of 250m.

The clock rates are randomly selected from the range [0.9999, 1.0001], following the IEEE 802.11 specification (cite). So starting from synchronized clocks, the maximum clock error between two nodes after 1 second is 200μs. Also, the initial clock value for each node is randomly chosen from the range [0, 1000] milliseconds. The beacon period is 100 milliseconds, unless otherwise specified.

Under these assumptions, we vary the packet loss rate to see the impact on the performance. If the packet loss rate is p, then a node transmits a packet, its neighbor successfully receives the packet with probability $1 - p$. Each receiver follows the fixed packet loss rate independent of other receivers.

Finally, the total simulation time is 1000 seconds for every simulations.

B. Simulation Results

Now we present and discuss our simulation results. As mentioned before, we evaluate our proposed protocol, MTSF, as well as IEEE 802.11 TSF with different probability that a node is forced to transmit beacon in a beacon period.

In the first simulations, we plot the global clock error over the whole simulation time. 100 nodes are randomly placed in 1000m $\times$ 1000m area, with clock rates and initial clock values randomly assigned.

Figure 6 shows the result for different protocols. Figure 7 plots the same graph, but with a smaller scale on the Y-axis. We can see that with MTSF, the maximum clock error is always bounded under a certain threshold. With the original IEEE 802.11 TSF, the clocks may drift away until the fastest node gets a chance to transmit the beacon. With increased probability of forced transmission, the accuracy of TSF increases, and the accuracy of TSF with forced transmission probability 0.4 is comparable to the accuracy of MTSF. We will see later that the number of packets transmitted is much smaller with MTSF.

If the global clock error exceeds a given threshold, we say that the network has become unsynchronized. We measure the percentage of time that the network is unsynchronized, for different thresholds. Figure 8 also shows that the accuracy that MTSF achieves is comparable to TSF with forced transmission probability between 0.2 and 0.4.

In addition to performance in terms of accuracy, we want to see how fast the network converges to a synchronized state using MTSF. Note that the initial values are assigned randomly from the range [0, 1000]. So initially, the clocks are unsynchro-

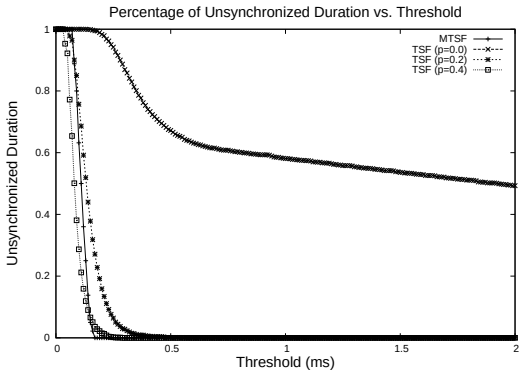


Fig. 8. Percentage of Time the Network is Unsynchronized.

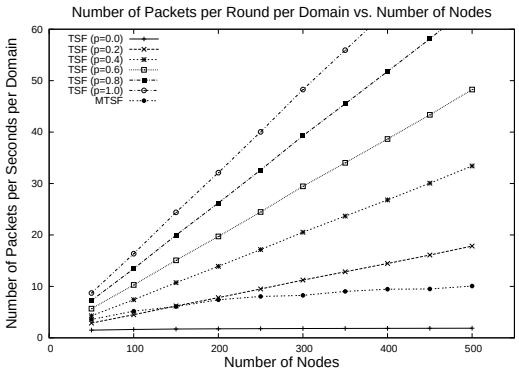


Fig. 10. Average Number of Messages Per Second Per Domain.

nized by the maximum of 1 second.

Figure 9 plots the global clock error over time with different protocols, but it only shows the initial part where the clocks are being synchronized. For IEEE 802.11 TSF, the convergence time is longer when the forced transmission probability is low. As the probability increases, the convergence time decreases dramatically. For MTSF, the convergence time is comparable to TSF with forced transmission probability 0.2. The overhead of MTSF is lower than that of TSF with probability 0.2, as seen later.

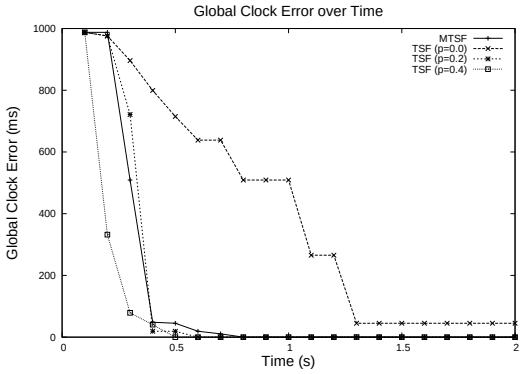


Fig. 9. Global Clock Error over Time.

Next, we study the overhead of the protocols. To see the communication overhead, we measure the average number of beacons sent or received per round in a broadcast domain.

Figure 10 shows the result. The message overhead of MTSF is less than TSF with forced transmission probability 0.2. Moreover, as the number of nodes increase, the increase rate of message overhead is much slower than TSF. This result indicates that MTSF is scalable.

The reduction in communication cost comes from building an efficient spanning tree in the network, with the fastest node as the root. If the percentage of leaf nodes is higher, then the communication cost is lower. To see how MTSF does well in terms of building a spanning tree, we measure the percentage of leaf nodes in a converged network. As shown in Figure 11(a), the percentage of leaf nodes increases as the number of nodes increase. Also, Figure 11(b) shows that the number of non-leaf nodes increases slowly as the number of nodes increases.

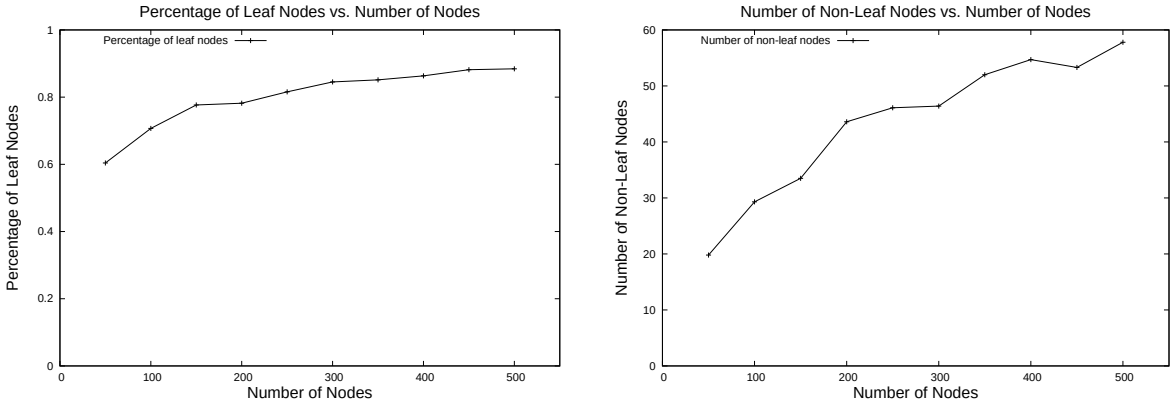
Until now, we assume that there is no packet loss. However, in reality, there can be packet loss due to bad channel quality, or packet collisions. In the next simulations, we study how MTSF performs under different packet loss rates. Figure 12 and Figure 13 shows the degradation in accuracy of MTSF when there is packet loss. The results show that MTSF tolerates packet loss well with only a slightly increased global clock error. This tolerance of MTSF comes from the redundancy in which a node may receive multiple clocks from neighbors in a round and synchronize to them. So even when the beacon from the parent is lost, there may be other neighbors that transmit beacons in the same round. If the beacon holds a faster clock, the node can synchronize to these beacons. When the network is not converged to a steady state, packet loss may lead to a longer convergence time because a node may temporarily switch its root. But once the synchronization tree is stabilized and all nodes have the same root, the redundancy improves the tolerance of the protocol to packet loss.

From the simulations, we have seen that MTSF achieves good accuracy even under packet losses, and the accuracy is achieved at a much lower cost than IEEE 802.11 TSF with forced transmission probability tuned to match the accuracy with MTSF. Also, we have seen that MTSF is scalable, because the overhead of MTSF increases slow as the node density increases.

VII. Conclusion

In this paper, we have proposed MTSF, a time synchronization protocol for multi-hop wireless networks. Since MTSF is designed to support synchronous operations for applications or other protocols running on top of MTSF, it aims to achieve stability in maintaining clock accuracy. At the same time, MTSF aims to reduce the communication cost used for synchronization. Reducing cost is important in achieving scalability.

Since a node only synchronizes to a faster node, all the nodes must synchronize with the fastest node in the network to avoid fastest node asynchronism. MTSF achieves this by implicitly building up a synchronization tree rooted at the fastest node in the network. During a short period of time when there is no change in network topology, the protocol quickly converges to a steady state. In the absence of packet loss, the protocol in the steady state guarantees an upper bound on the global clock error. In the presence of packet loss, the protocol tolerates when the loss rate is low, and the performance degrades gracefully as



(a) Percentage of Leaf Nodes (b) Number of Non-leaf Nodes

Fig. 11. The leaf nodes in the synchronization tree built by MTSF protocol.

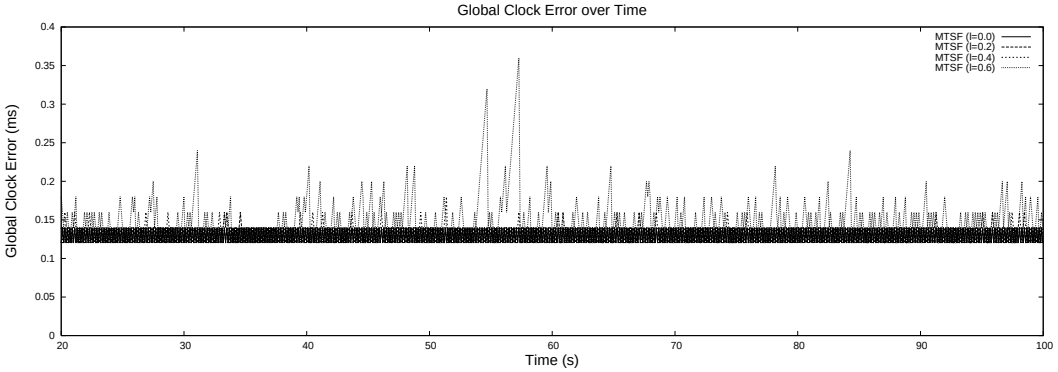


Fig. 12. Global Clock Error over Time.

the loss rate becomes high. When the topology changes, the protocol quickly self-stabilizes to another steady state, without any explicit procedures. We have proven that the network converges to a steady state once the network topology is fixed, and also calculated the upper bound on the global clock error in the steady state.

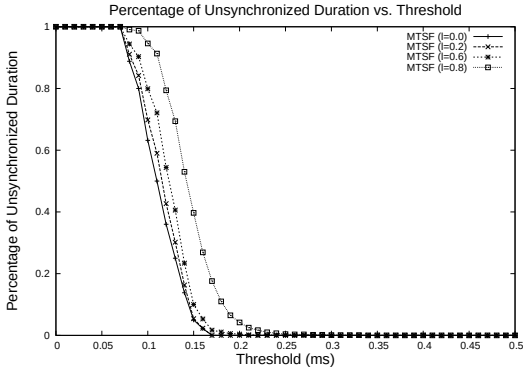


Fig. 13. Percentage of Time the Network is Unsynchronized.

Also, using the tree structure, MTSF reduces the number of beacon transmissions in each period. This is important because the synchronization process should not harm the performance of applications that use the synchronization service, by occupying significant amount of bandwidth.

The simulation results show that MTSF achieves stable clock accuracy at a low cost. So MTSF can efficiently support synchronous operations which is an important requirement for many applications and protocols. We are planning to implement MTSF with other protocols that use synchronous operations, to test the effectiveness of MTSF in supporting synchronous operations. The results will be reported in the near future.

REFERENCES

[1] IEEE 802.11 Working Group, "Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications," 1997.
[2] L. Huang and T. Lai, "On the Scalability of IEEE 802.11 Ad Hoc Networks," in *ACM MOBIHOC*, June 2002.
[3] Y.C. Tseng, C.S. Hsu and T.Y. Hsieh, "Power-Saving Protocols for IEEE 802.11-Based Multi-Hop Ad Hoc Networks," in *IEEE INFOCOM*, 2002.
[4] I.A. Getting, "The Global Positioning System," *IEEE Spectrum 30*, December 1993.
[5] title = C. Schurgers, V. Tsiatsis, S. Ganeriwal and M. Srivastava, ," .
[6] D. Mills, "Internet Time Synchronization: The Network Time Protocol," *IEEE Trans. Communications.*, October 1991.
[7] J. Elson and D. Estrin, "Time Synchronization for Wireless Sensor Networks," in *International Parallel and Distributed Processing Symposium (IPDPS)*, April 2001.
[8] J. Elson, L. Girod, and D. Estrin, "Fine-Grained Network Time Synchronization using Reference Broadcasts," in *Fifth Symposium on Operating Systems Design and Implementation (OSDI)*, December 2002.
[9] Kay Römer, "Time Synchronization in Ad Hoc Networks," in *ACM MOBIHOC*, October 2001.

[10] M.L. Sichitiu and C. Veerarittiphan, "Simple, Accurate Time Synchronization for Wireless Sensor Networks," in *IEEE Wireless Communications and Networking Conference (WCNC)*, 2003.
[11] Jana van Greunen and Jan Rabaey, "Lightweight Time Synchronization for Sensor Networks," in *Workshop on Wireless Sensor Networks and Applications (WSNA)*, September 2003.
[12] S. Ganeriwal, R. Kumar, M. B. Srivastava, "Timing-sync Protocol for Sensor Networks," in *First ACM Conference on Embedded Networked Sensor Systems (SenSys)*, November 2003.
[13] Q. Li and D. Rus, "Global Clock Synchronization in Sensor Networks," in *IEEE INFOCOM*, 2004.
[14] J.-P. Sheu, C.-M. and C.-W. Sun, "A Clock Synchronization Algorithm for Multi-Hop Wireless Ad Hoc Networks," in *IEEE ICDCS*, 2004.