A Distributed Self-Stabilizing Time Synchronization Protocol for Multi-hop Wireless Networks [*]

Technical Report

January 2004

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

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

ABSTRACT

In this paper, we propose a time synchronization protocol for multi-hop wireless networks. The *Multi-hop Time Synchronization Protocol* (MTSP) is designed to work in multi-hop wireless networks, with the following assumptions. First, a node can only synchronize to another node with a faster clock to avoid clocks from going backwards. Second, the time of synchronization must also be synchronized, because nodes may not be able to participate in the synchronization process all the time. Nodes might go to sleep or listen to another channel after the synchronization process. Under these assumptions, the goal of the synchronization protocol is to maintain the *network synchronization error*, which is the maximum clock difference between any pair of nodes in the network, small. These requirements are critical for power management schemes, multi-channel protocols, and other protocols that need clock synchronization. Other proposed synchronization protocols do not work well under these requirements.

MTSP is fully distributed and self-stabilizing. Starting from an arbitrary state where each node is at arbitrary time, MTSP makes the network converge to a steady state. In the steady state, a spanning tree is established, rooted at the fastest node. If there is no packet loss, MTSP guarantees an upper bound on the synchronization error in the steady state. If the topology changes due to mobility, the protocol self-stabilizes to a steady state soon after the network becomes static again. We show that the network self-stabilizes to a steady state and calculate the upper bound on the network synchronization error in the steady state. Also, we show by simulations that MTSP can achieve reasonable accuracy at the cost of low overhead. Finally, we study the impact of controlling transmission power for time synchronization, using MTSP.

[*]This research was supported in part by Motorola.

1. INTRODUCTION

Time synchronization is an important feature in distributed systems. Many applications and protocols require time synchronization among nodes, with various requirements on the amount of tolerable error. One example of a protocol which requires synchronization is the power saving mechanism (PSM) of IEEE 802.11 [3]. In IEEE 802.11 PSM, all nodes must wake up at the beginning of a beacon interval to exchange messages. While they are awake, they exchange messages to decide whether to sleep or not for the rest of beacon interval. If a node decides to sleep during the beacon interval, it turns off its radio and goes to sleep after the message exchange process is finished. Another possible application of clock synchronization is a multi-channel protocol using synchronized channel negotiation. In this protocol, nodes listen to different channels most of the time. Periodically, all nodes switch to a common channel to negotiate channels by exchanging messages. After negotiating the channels, the nodes switch to different channels according to their selection during the negotiation phase. To make this scheme work, the clocks need to be synchronized.

There are two major reasons that we need time synchronization among nodes. The first reason is to figure out the timing of events. Many applications of sensor networks rely on time synchronization to obtain useful information from sensed data. For example, suppose the nodes in Figure 1 are sensors deployed on a road to detect car movement.

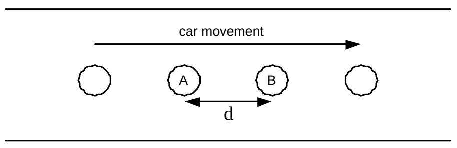


Figure 1: Sensors deployed to detect car movement.

Suppose a car movement was detected by the sensors. Node A reports that the car was detected at 1 o'clock, and node B reports that the car was detected at 1 second after 1 o'clock. We know that node B is located 10m east of A, and we want to find out the direction and the velocity of the car. Can we conclude that the car is moving east at a speed of 10m/s? We cannot, because the clocks at node A and B may not be synchronized. if the clock of node B was running 2 seconds faster than A, the car might have actually been moving west.

We need clock synchronization for this kind of applications, but the clocks need not be synchronized at the moment the event happens. We can just leave the nodes unsynchronized, and then figure out the timing of events considering the clock difference later on. This approach for time synchronization is called post-facto synchronization, proposed by Elson and Estrin [4].

Another reason that we need time synchronization is to have multiple nodes generate synchronized events. In a synchronized power management scheme, the nodes must wake up at the same time to exchange messages. In a synchronized channel allocation algorithm, the nodes must listen to a common channel at the same time to negotiate channels. For these applications, the clocks must be synchronized prior to the event, so that the events happen at the same time. Suppose we want to have two sleeping nodes wake up at the same time and communicate with each other. If the time is not synchronized, each node will wake up at a different time. In this case the receiver might not receive a packet just because its clock was running slower and it did not wake up by the time the sender sent the packet. Since we do not know which two nodes in the network is going to communicate with each other, we need to synchronize the whole network.

In this paper, we focus on the latter case. The post-facto synchronization does not work in this situation, and we need *a priori* synchronization to main the whole network always synchronized. To maintain the network synchronized, nodes need to exchange messages from time to time to synchronize their clocks with each other and prevent clocks from drifting away.

Tseng et al. points out the difficulty of using IEEE 802.11 PSM in multi-hop networks [14]. The three main problems are clock synchronization, neighbor discovery, and network partitioning. These problems occur because all of the nodes can periodically go to sleep and wake up, according to their schedules. So even if two partitioned network move close to each other, they may not discover each other because the nodes of two groups are awake at different times. To avoid this problem, we require that a subset of nodes are always able to receive synchronization packets. Then the nodes can eventually discover all the neighbors, and synchronize to each other. One example of protocol that shows this kind of behavior is SPAN [1].

There are several issues to consider in the environment of our interest. First, we assume that there is no predetermined reference clock in the network. In wired networks, the Network Time Protocol (NTP) [2] uses reference clocks to synchronize the nodes. However, we cannot assume such a reference clock exists in wireless ad hoc networks. So the goal of a time synchronization protocol in this environment is to keep the time difference between any pair of nodes in the network small. We cannot just pick any node in the network and use its clock as a reference clock, because of the following problem. One rule in time synchronization is that a node only synchronize to faster node. Note that when we say node A is *faster* than node B, this means that node A has a clock value larger than node B at some point of time. The reason that a node only synchronizes to a faster node is to prevent local clocks from going backwards, which is considered harmful. Thus, if we pick any node in the network and and use its clock as a reference clock, nodes with a faster clock may drift away [8]. To prevent this problem, nodes must synchronize to the fastest node in the network.

Second, nodes might not always be available to participate in the synchronization process. In a power management scheme, all nodes are awake for only a certain duration of time between the synchronization interval, and nodes may go to sleep after that duration. Since a node cannot participate in the synchronization process while it is asleep, the synchronization process must take place while all nodes are awake. The same situation happens in a multi-channel protocol. Once nodes switch to different channels, they cannot participate in the synchronization process. Thus, the synchronization process must also be synchronized.

To summarize, the environment model we consider has the following assumptions.

- We consider a wireless ad hoc network with no infrastructure support. Also, there is no reference clock in the network.
- A node only synchronizes to another node with a faster clock. Thus, all nodes must synchronize to the fastest node to prevent the fastest node from drifting away.
- The synchronization process must be synchronized, because nodes may not be available to participate in the synchronization process all the time.

Under these assumptions, the goal is to make the *network synchronization error* small. The network synchronization error is defined as the maximum of time difference between any pair of nodes in the network. Also, it is desirable to achieve this goal at the cost of low communication overhead.

Our proposed protocol, called *Multi-hop Time Synchronization Protocol* (MTSP), is designed to work in this environment. MTSP establishes a synchronization tree rooted at the fastest node. The tree is not obtained using an explicit exchange of messages, but the protocol eventually converges to a state where the synchronization tree is established. The protocol is self-stabilizing, meaning that starting from an arbitrary state where each node has arbitrary logical time, it eventually enters a state where the synchronization tree rooted at the fastest nodde is established and the clocks are synchronized within a certain amount of error. If we assume that there is no packet loss, the protocol guarantees an upper bound on the time difference between any pair of nodes in the network. The protocol is fully distributed, and can adapt to the dynamics of the network. So when a new node which has the fastest clock joins in, the protocol automatically switches to forming a tree rooted at the new node.

The outline of this paper is as follows. In section 2, we define the problem we are addressing formally. In section 3, we review the work done in the area of clock synchronization. After that, we describe our proposed protocol, MTSP, in section 4. In section 5, we evaluate the performance of MTSP using simulations. Finally we draw our conclusion in section 6.

2. PROBLEM DEFINITION

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

We consider a wireless ad hoc network that consists of multiple nodes connected to each other through wireless interface. Infrastructure does not exist, and the network may span multiple hops.

Each node maintains a hardware clock. We assume that all hardware clocks have the same granularity of representing the time. 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. We denote the real time as t. Then, 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 and β 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, called the *logical clock*. Logical clock is a representation of the hardware clock, but it can be modified by the system. 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 is the parameter that can be controlled by the operating system. A node can correct the logical time to reduce the time difference with other node, and this process is called the *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, we will only consider the logical time and not use physical time. So when we say the "time" of a node, we mean logical time.

Now the goal of a synchronization protocol is to adjust the δ value of each node so that the time difference among nodes is kept small. In order to synchronize the time, nodes must exchange messages telling their local time to other nodes. Because of the uncertainty in message delay, the network cannot be synchronized to the exact same time. Every pair of nodes may have a time difference, and we denote the time difference between node i and j as Δ_{ij}.

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

The goal is to make the network synchronization error small so that it can be kept below a certain threshold for most of the time. The threshold value represents the amount of accuracy an application requires. If we represent the network as a graph (V, E), the goal of a synchronization protocol is to meet the following requirement:

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

for all i,$j \in V$. W is called the *synchronization threshold.*

If the protocol guarantees that the above equation holds for most of the time, the application that makes use of clock synchronization can use this information to set up a margin before starting a synchronized state. 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 2.

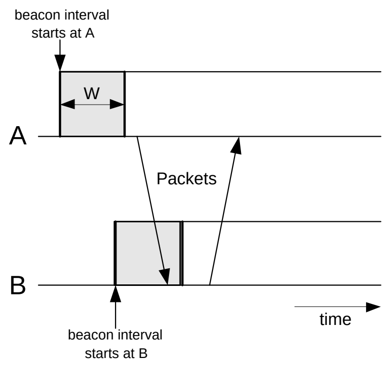


Figure 2: Node A knows that the maximum synchronization error is W, so A waits for W at the beginning of its beacon interval and starts transmitting.

3. RELATED WORK

Numerous time synchronization schemes exist in the context of wired and wireless networks [2, 4, 5, 7, 9, 6, 10]. Traditionally in wired networks, the main focus of a time synchronization is to reduce the network synchronization error as much as possible, without much concern on message overhead. Also, synchronization protocols for wired networks are designed with the assumption that the network is mostly static, although they have some fault tolerant features to overcome node and link failures. Recently, several time synchronization protocols for wireless networks have been proposed. These protocols consider the dynamic nature of wireless networks. Also, time synchronization protocols designed for wireless sensor networks have much concern on energy consumption, because energy is a critical factor in sensor networks. In this section, we review the existing work in the area of time synchronization.

In wired networks, clock synchronization is achieved using the Network Time Protocol (NTP) [2]. 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.

However, NTP does not work well in wireless ad hoc networks, where topology may change frequently due to mobility. This is because NTP's synchronization hierarchy is pre-configured, assuming the network is static. To synchronize nodes in a wireless ad hoc networks, the protocol must adapt to link failures, topology changes and even temporary partitions of the network. Also, in wireless ad-hoc networks, reference clocks may not be available. In this kind of environment, the goal of time synchronization is to keep the time difference between any two nodes in the network as small as possible.

Now we review the synchronization protocols for wireless networks. IEEE 802.11 standard has a time synchronization function (TSF) designed for a wireless LAN [3]. The protocol is based on beacon transmission. At the beginning of each beacon interval, each node picks a random delay before transmitting the beacon packet. If a node transmits a beacon, all other nodes in the network will receive the beacon. When a node receives a beacon, it suppresses its beacon, and synchronizes to the node that sent the beacon, using the timestamp included in the beacon packet. Huang and Lai [8] identifies the problem of this scheme, which is the *fastest node asynchronism*. A node only synchronizes to another node which has a faster clock then itself. This is to avoid moving backward in time, which may ruin the local ordering of events. Thus the fastest node will keep drifting away from other nodes, unless it becomes the beacon transmitter. Huang and Lai propose a simple modification to reduce the impact of fastest node asynchronism. This scheme works well in wireless LAN. However, when these schemes are applied to multi-hop networks, the clocks might still drift away because of the *time clustering* problem. The time clustering problem will be explained in the next section.

One major challenge in synchronizing two nodes is estimating the packet delay between the sender and the receiver. This is mainly due to contention-based medium access, and the delay before a node gains the channel access varies dramatically depending on the amount of traffic and channel condition. This delay is called *access time*, and it is the major source of error in estimating the delay of packet delivery.

To address this issue of uncertainty in the access time, Elson et al. proposed Reference-Broadcast Synchronization (RBS) [5]. Instead of sender-receiver synchronization, this protocol performs receiver-receiver synchronization. The basic idea of RBS is to have a reference node broadcast a reference packet, and multiple receivers synchronize to each other comparing the packet arrival time of the reference packet. In this approach, the uncertainty of *access time* is removed from and the accuracy is increased. However, the overhead of RBS is very high, because each node must receive a message from the reference clock, and exchange messages with each other to synchronize their clocks.

Ganeriwal et al. proposed a scheme similar to NTP, but a modified version that works in sensor networks [10]. The protocol, called Timing-Sync protocol for Sensor networks (TPSN), uses pairwise message exchange to remove the uncertainty in the message delay. The pairwise message exchange does not remove the uncertainty of access time, but they remove it by stamping time on the packet at the MAC layer, just before transmitting the packet. To synchronize a multi-hop network, the protocol establishes a hierarchical structure in the initial level discovery phase. After that, the protocol enters the synchronization phase, where pairwise synchronization is performed along the edges of the tree. This protocol assumes a pre-assigned root node, and does not address the issue of *fastest node asynchronism.* Also, $2n$ transmissions are required for each synchronization process, which is a higher overhead, especially in a dense network.

The lightweight tree-based synchronization algorithm (LTS) [6] proposed by Greunen and Rabaey, also uses pairwise message exchange for synchronizing two nodes. LTS has centralized and distributed version for multi-hop synchronization, and the centralized version is similar to TPSN. In the distributed version, each node chooses the synchronization period based on the desired accuracy. When a node needs to be synchronized with other nodes, it sends a request packet to the reference node to trigger the synchronization process. Then all the nodes in the path to the root node will be synchronized using pairwise communications. The main advantage of LTS over TPSN is that LTS may have less overhead, because nodes that do not require frequent synchronization are synchronized at a slow rate. Although LTS can be efficient then TPSN, but it still has the problem of fastest node asynchronism.

The above two protocols use pair-wise message exchange to remove the error of estimating delays. However, if we remove access time by timestamping at the MAC layer similar to [10], a node can estimate the delay with sufficient accuracy, using only one packet sent by the sender. The process of synchronizing a pair of nodes is explained in the next section.

Elson and Estrin proposed the concept of *post-facto synchronization* [4]. 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 and a priori synchronization have the characteristics of reactive and proactive protocols, respectively. If events do not occur frequently and only a small portion of the networks is relevant to the event, then post-facto synchronization is more efficient than a priori synchronization. On the other hand, if events occur frequently and span over most of the network, a priori synchronization may be more efficient.

Our proposed protocol, MTSP, takes the approach of a priori synchronization, because post-facto synchronization is not suitable for the environment of our interest. Post-facto synchronization is only useful when the purpose is to determine the time at which the events happened, but cannot be used to coordinate node activities such as synchronous wake up.

4. PROTOCOL DESCRIPTION

In this section we present the details of our proposed protocol, the *Multi-hop Time Synchronization Protocol* (MTSP). First we describe the method used to synchronize a pair of nodes. Then we discuss how we can synchronize the whole network which spans multiple hops. As a conclusion of the discussion, we present MTSP.

4.1 Synchronizing a pair of nodes

To synchronize a pair of nodes A and B, we need to have either A or B send a message, including the timestamp generated by its logical clock. Suppose node A generates a packet and stamps the time at t_1. The timestamp value is the logical time of node A at t_1, which is $T_A(t_1)$. After stamping the time, node A sends the packet to B. When node B receives the packet, it also stamps the time using its logical clock to record the time of reception. Suppose at t_2, node B stamps the time. Now node B knows two values: $T_A(t_1)$ and $T_B(t_2)$. To find out the time difference, node B needs to estimate the delay from t_1 to t_2, so that it can obtain an estimated value of $T_A(t_2)$. The time difference between two nodes at t_2 is $|T_A(t_2) - T_B(t_2)|$. Once B determines the time difference between A and B, it can synchronize to A by adjusting the clock offset δ in its logical clock. Note that node B can synchronize to node A only if A's clock is running faster than B's.

The delay between t_1 and t_2 is called the *critical path* [5]. The reason that B cannot exactly synchronize to A is due to the uncertainty in the critical path. The critical path can be decomposed into five components as follows. The decomposition follows from [5, 10].

- Send time: This is the delay from the time a packet is generated until the time it reaches the MAC layer for transmission. Send time mainly depends on the operating system of the sender.
- Access time: After the packet reaches the MAC layer, it has to wait until the node gains access to the channel. Access time is the delay between the time a packet reaches the MAC layer and the time the node starts transmitting the packet. This delay depends on the MAC protocol the network uses.
- Transmission time: When the node gains access to the channel, the node transmits the packet, bit by bit. Transmission time is the duration from the time the first bit of the packet is transmitted, until the time the last bit is transmitted. This delay is a function of transmission rate and packet size.
- Propagation time: This is the duration for a packet to travel from the source to the destination. Propagation time is a function of distance between the source and the destination.
- Receive time: This is the delay from the time when the packet reaches the destination node until the time the destination node stamps the time to record the time of reception. Receive time depends on the operating system of the receiver.

The components of the critical path are illustrated in Figure 3. The major source of error among these components is the access time, because most common MAC protocols are contention-based, and the access time varies significantly based on the channel condition at the time of packet transmission. The send time and the receive time may also be significant depending on the operating system policy. Compared to the other components, the transmission time and the propagation time are insignificant in their contribution to the estimation error.

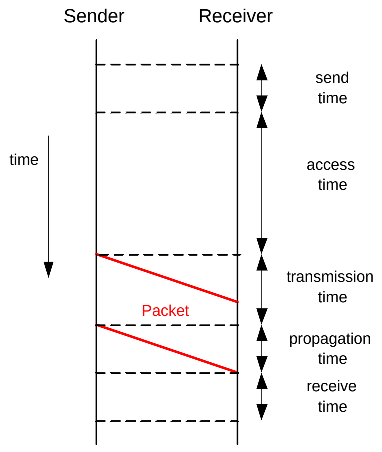


Figure 3: Components of the critical path.

We can significantly reduce the estimation error using the following two strategies. First, the sender stamps the time at the MAC layer, just before putting the packet on the channel. This strategy is also used in [10]. This removes the send time and the access time from the critical path. Second, the receiver records the time of reception as soon as it starts receiving the packet. This removes the receive time from the critical path.

Now only the transmission time and the propagation time are left. 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 the transmission rate and the packet size, we can accurately estimate this delay in unit of real time. However, the rate of the receiver's clock is different from the real clock, and this is where the estimation error occurs. Since the clock rate at the receiver is α_R, the error of 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 $\pm 0.01\%$. We follow this specification and assume that the clock rates are within the range [0.9999, 1.0001]. Also, we assume that the synchronization packet size is 56 bytes, which is the size we use in our simulations. The 56 bytes consist of 24 bytes of preamble, which is transmitted at 1Mbps, and other 32 bytes which can be transmitted at the transmission rate of 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 time, we need to know the distance from the source to the destination. However, since the propagation time is very small, we can just use an upper bound on the propagation time and tolerate the error. Then the maximum estimation error for the propagation time would be:

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

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 0.8us. To reduce the error for propagation time, the receiver can measure the signal strength of the received packet and estimate the distance from the sender. However, we use a fixed propagation delay in our simulations.

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, we can synchronize a pair of nodes with accuracy less than 1μs.

4.2 Synchronizing the network

Now that we know how to synchronize a pair of nodes, we want to synchronize the whole network, meaning that we want to make the clock difference of any two nodes in the network small. We first discuss several alternative approaches to synchronizing the network, then present proposed MTSP protocol in the next section.

Before we discuss the alternative approaches, we summarize the requirements again. First, a node only synchronizes to a faster node. So every node must be synchronized to the fastest node in the network. Second, The synchronization process must happen at a synchronized period of time. This is because nodes might not be able to participate in the synchronization process all the time. The nodes might be sleeping, or listening to different channels. At a synchronized point of time, they all wake up or listen to a common channel for the purpose of synchronization.

The most simple way to synchronize the network is to have one node, say node R, flood the whole network with its beacon periodically. The beacon transmitter node can be preassigned, or elected using a leader election algorithm. There are several problems that make this scheme not work well under our requirements. First, node R may not be the fastest node. Since a node only synchronizes to a faster node, nodes that are running faster than R will keep on drifting away. Second, the duration of the synchronization process, which we call the *synchronization period*, may be very long, since each node has to wait for a beacon to come and then rebroadcast it[1]. A node that is far from node R must wait a long time to receive a beacon, after the beacon interval starts. Consider the chain topology shown in Figure 4. Suppose node 1 is assigned to be the beacon transmitter. We measured the average synchronization period varying the distance d between nodes. As shown in Figure 5, the synchronization period can be very long when flooding algorithm is used. A long synchronization period degrades the performance of the protocol, which is not desirable. Thus, flooding-based schemes are not suitable for our environment model. In Figure 5, the average synchronization period increases as the distance between two adjacent nodes increases. This is because the maximum hop distance from node 1 increases. If the distance between adjacent nodes becomes more than 150m, the synchronization period does not increase any more because the maximum hop distance does not change. Since the transmission range of a node is 250m, node i can only reach node $i-1$ and $i+1$, when the distance between adjacent nodes exceeds 150m.

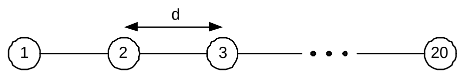


Figure 4: A chain topology with 20 nodes. The distance between two adjacent nodes is fixed.

[1]In a power management scheme, a node may wake up just for the purpose of synchronization, even if it does not have packets to send or receive. In a multi-channel MAC protocol, a node switches to the common channel periodically to be synchronized with other nodes. In both cases, the synchronization period is an overhead, so it is desirable to have short synchronization period

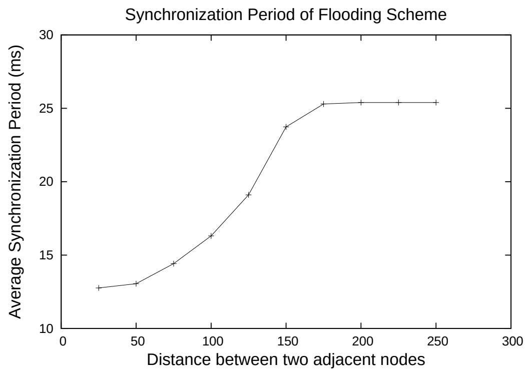


Figure 5: The average synchronization period of flooding scheme.

Instead of one node flooding the network, each node can independently transmit beacons to synchronize with its neighbors. For a wireless LAN, IEEE 802.11 Timing Synchronization Function (TSF) [3] may be used to synchronize the network. At the beginning of each interval, every node waits for a random delay before transmitting a beacon. If a node receives a beacon from another node during this delay, it suppresses its own beacon, and synchronizes to the sender if the sender is running faster than the receiver. We can apply this scheme to a multi-hop network. We refer to this scheme as "802.11 TSF". Using this scheme, only one transmission occurs in a "broadcast region". So this scheme has short synchronization period and low overhead. However, as identified in [8], the clock of the fastest node may drift away, because the fastest node may not get a chance to transmit its beacon. Since the fastest node does not synchronize to any other node, its clock will keep drifting away from other clocks. This problem is called *fastest node asynchronism* [8].

Another problem of using 802.11 TSF in a multi-hop network is *time clustering*. Consider the scenario in figure 6. If each node waits for a random delay and transmits the beacon, it might happen that node A always sends beacon before B and node D always sends beacon before C. Then the nodes form clusters, A and B in one cluster, and C and D in the other cluster. The clocks in these clusters may drift away unboundedly. Due to this time clustering problem, 802.11 TSF cannot guarantee that the network is always synchronized within a certain bound.

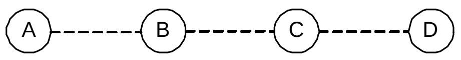


Figure 6: A simple chain topology with 4 nodes.

To avoid the fastest node asynchronism and time clustering, we can have every node transmit beacon at each beacon interval. We refer to this scheme as "One-Hop Broadcast". Since every node transmits beacon, the fastest node will also transmit beacon and the other nodes can synchronize to the fastest node. Also, since a node receives beacons from all its neighbors, the time clustering problem cannot happen. However, the cost of this scheme is very high, especially for a dense network, since n transmissions take place every beacon interval.

MTSP follows the approach of One-Hop Broadcast scheme, but reduces the number of beacon transmission in one interval by establishing a synchronization tree rooted at the fastest node. The tree structure makes every node in the network synchronized with the fastest node, by having only a subset of nodes transmit beacons at each beacon interval. The details of MTSP is explained in the next section.

4.3 Multi-hop Time Synchronization Protocol (MTSP)

The main idea of MTSP is to establish a synchronization tree in the network, rooted at the fastest node. The desired synchronization tree is such that the depth of the tree is minimized, and the number of leaf nodes is maximized. The depth of the synchronization tree determines the upper bound of network accuracy, and the number of non-leaf nodes affects the overhead of the protocol.

First, we look at how MTSP establishes a tree rooted at the fastest node. Unlike TPSN [10], MTSP does not need an explicit process to form a synchronization tree. Instead, each node independently picks one node among its neighbor as its parent node. It can be shown that if a node chooses as parent a node which has the fastest clock among neighbors, the node will eventually point towards the fastest node, meaning that it will join the tree rooted at the fastest node. So when every node always picks its parent this way, the whole network eventually converges to forming a synchronization tree rooted at the fastest node. So MTSP is self-stabilizing, meaning that starting from an arbitrary state where each node has arbitrary logical time, the protocol eventually enters a state where the synchronization tree rooted at the fastest node is established. We state this in Theorem 1, and present the proof in Appendix A.

THEOREM 1. *Starting from an arbitrary state, the protocol will enter a steady state, where a synchronization tree rooted at the fastest node is established in the network, provided that the network is steady for the duration of stabilization. Thus the protocol is self-stabilizing.*

Next, we look at how MTSP reduces communication overhead. The idea is to establish the tree, which has a large number of leaf nodes. A leaf node in the synchronization tree is a node that has a parent, and does not have any children. The leaf nodes do not need to transmit beacons every interval, because no node benefits from their transmissions. However, if leaf nodes do not transmit beacons at all, two partitioned networks may not be able to find each other even if their leaf nodes are adjacent to each other. So we want to have only a subset of leaf nodes transmit beacons at each beacon interval. To achieve this, we apply the counter-based method from broadcast storm solutions [11]. With this method, in each beacon interval, if a leaf node receives more than m beacon packets before transmitting its own beacon, it suppresses its beacon. m is called the *suppress threshold.*

However, this method may result in time clustering, if it is used without caution. Suppose node A is a leaf node of one tree and node B is a leaf node of another tree. These two trees are initially partitioned, because they are distant with each other. Later on, the two trees move closer to each other, so that B and D become close enough to reach each other. However, it is possible B and D might never transmit beacons because their beacons are suppressed by the beacon suppression method in every beacon interval. In this case, the two network cannot be merged in terms of clocks. To prevent this, it must be guaranteed that every node will eventually transmit beacon. To implement this idea, we still assign a non-zero probability to a leaf node, which receives m beacon packets in a beacon interval. The probability is proportional to the *number of consecutive beacon intervals the node has been silent.* After a fixed number of beacon intervals, the probability becomes 1, so that the node will be forced to transmit a beacon. This guarantees that a node will eventually find all of its neighbors, and thus it prevents time clustering in the network.

A node may have multiple potential parents that points towards the same root. If two potential parents point towards the same root, the node can choose either one as its parent, without choosing the faster one (The purpose of choosing a faster node is to point to the fastest node). In this case, nodes can choose a parent based on other criteria. To make the depth of the synchronization tree as small as possible, it is preferable for a node to choose the parent with the least hop count from the root. On the other hand, To make the number of leaf nodes large, it is preferable for a node to choose a parent which has the largest number of children. We give priority to minimizing the depth of the synchronization tree. So among multiple potential parents, a node chooses the one with the least hop distance from the root. If there are multiple nodes that have the same hop count, then a parent with the largest number of children is chosen.

Now we describe the protocol in detail.

In MTSP, each node maintains four variables: p, r, d, and c: p is the address of the parent node, and r is the address of the root node in the synchronization tree. d is the hop distance from the root node, and c is the number of children the node has in the synchronization tree. In addition to these variables, each node maintains the list of children, which is denoted as C. So c is the number of elements in C. Initially in every node, p is NULL, r is the address of itself, d is infinity, c is zero and C is empty.

If a node has a parent and no children, it regards itself as a *leaf node*, otherwise it is a *non-leaf node*. Initially, all nodes are non-leaf nodes because any node does not have a parent.

At the start of a beacon interval, each node picks a random number from the range [0, BCW] and waits for the selected number of "slots" before transmitting the beacon. BCW is the *beacon contention window*, and we use 62 in our simulations. If the node is a non-leaf node, it transmits the beacon after the delay, regardless of how many beacons it receives while waiting to access the channel. If the node is a leaf node, it counts the number of beacons it receives while waiting to access the channel. If m beacon packets are received during the wait, the node suppresses its own beacon. If less than m beacon packets are received, the node transmits its beacon. The leaf node also keeps track of how many consecutive beacon intervals it did not transmit beacon. If a node has not transmitted beacon for b consecutive beacon intervals, it transmits beacon with propability b/B even though it receives m beacon packets in the current beacon interval (B is a constant). Thus, a node is forced to transmit a beacon at least every B beacon intervals. This guarantees that a node will eventually find all of its neighbors, and thus it prevents time clustering in the network.

Suppose node A is transmitting the beacon. Then node A includes in the packet the four variables it maintains. p and r are 6 bytes each, and d and c are 2 bytes each. Thus we need 16 additional bytes to include the four variables in the beacon packet. When node A gains access to the channel, A adds a timestamp to the packet and transmits the packet.

Suppose node B receives a beacon from node A. We denote the variables of A as p_A, r_A, d_A and c_A, and similarly denote the variables of B as p_B, r_B, d_B and c_B.

When the beacon is received, B estimates the packet delay using the scheme explained in section 4.1, and obtains T_A. If $T_A > T_B$, then B synchronizes to A by adjusting its clock offset δ_B.

$$\delta_B \leftarrow \delta_B + T_A - T_B \tag{13}$$

If $T_A \leq T_B$, then B does not adjust its clock.

First we look at the case where A was running faster than B. After adjusting the clock, B has to decide whether it should regard A as its parent toward the fastest node. To do this, B compares r_B with r_A. If $r_B \neq r_A$, then B regards A as a new parent, and updates its variables as the following [2].

$$p_B \leftarrow A, \quad r_B \leftarrow r_A, \quad d_B \leftarrow d_A + 1 \tag{14}$$

If $r_B = r_A$ then B compares d_B with d_A. If $d_B > d_A+1$, then B can reduce the hop distance from the root by choosing A as its parent. So in this case, B chooses A as a new parent and updates its variables. If $d_B \leq d_A$ then B does not choose A as its parent.

If $d_B = d_A + 1$, this means that d_B does not change when B chooses A as its parent. In this case, B compares c_B with c_A. If $c_A > c_B$, then B chooses A as its parent and updates its variables. If $c_A \leq c_B$, then B does not choose A as its parent. This is to make the number of leaf nodes in the synchronization tree as large as possible.

Next we look at the case where A was running slower than B. In this case, B does not synchronize to A. Instead, B checks to see if A's parent is node B itself. If $p_A = B$, then B adds A in its list of children C, if A was not already in the list. If $p_A \neq B$, then B checks the list of children C, and removes A if A was previously a child of B. c is updated according to the number of elements in C.

[2]We show in Appendix A that if the node always chooses this way, it will eventually point towards the fastest node.

In Appendix A, we show that MTSP is self-stabilizing, by proving Theorem 1. Also, in Appendix B, we calculate the upper bound on the synchronization error in the steady state.

5. SIMULATIONS

In this section, we report the results from simulations we performed to study the performance of our protocol, MTSP. We also implemented IEEE 802.11 TSF and One-hop Broadcast scheme to compare with MTSP.

As stated earlier, we have two main goals. First, the clock difference between nodes in the network must be small. Second, the communication overhead must be low.

To study how the synchronization protocols achieve these goals, we measure the following metrics.

- Maximum Synchronization Error: This is the maximum of clock differences between any pair of nodes in the network, over the whole simulation time. The synchronization protocols that have probabilistic aspect such as IEEE 802.11 TSF, might have low synchronization error most of the time, but sometimes the clocks can drift away unboundedly, as explained in 4.2. On the other hand, MTSP aims to deterministically maintain the bound on synchronization error. Measuring the maximum synchronization error will capture this difference.
- Communication Overhead: This is measured as number of bits transmitted per second, because different protocols use different beacon sizes. This metric will show how successfully MTSP reduces the cost of synchronization.

After comparing the protocols, we study the impact of transmission power control on the performance of MTSP. Since the time of synchronization is also synchronized, we can assume that nodes use a different transmission power during the synchronization period, for the purpose of synchronization. If higher transmission power is used, the accuracy is expected to increase because the network diameter becomes smaller.

First we describe our simulation setup, and then we present the results.

5.1 Simulation Setup

We have used ns-2 [13] with CMU wireless extensions [12] for our simulations. For medium access control, CSMA/CA scheme is used since it is widely used for wireless communication.

We require the physical clock accuracy to be within $\pm 0.01\%$, which follows from the IEEE 802.11 specification [3]. So after 1 second, the difference between the fastest clock and the slowest clock is at most 20μs. The beacon period is 100ms, unless otherwise specified.

For MTSP protocol, we use 56 bytes for the beacon packet size. For other protocols, we use 40 bytes. This is because a beacon packet of MTSP requires 16 additional bytes to include more information, as explained in section 4.3.

If a node decides to transmit a beacon in the beacon period, it waits for a random delay before transmitting, to reduce the chance of collision. To make the random delay, each node picks a random value between [0, BCW]. We use 62 "slots" for the beacon contention window (BCW), which is also used in [8]. Each slot is 20 μs.

The transmission range of each node is 250m, unless otherwise specified. So the maximum propagation delay is approximately 0.83μs. When estimating the packet delay, we use a fixed estimated propagation delay of 0.5μs, which is roughly the half of maximum propagation delay.

Finally, the total simulation time is 100 seconds for each simulation, and each data point in the graphs is result of 20 simulation runs.

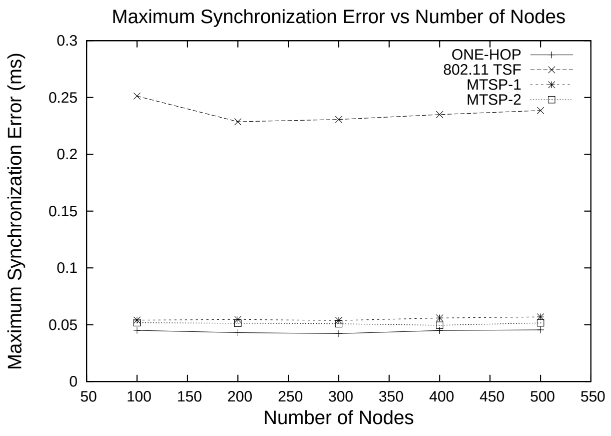
5.2 Simulation Results

Figure 7(a) and 7(b) shows the result from simulations performed in a 1000m $\times$ 1000m area. The number of nodes is varied from 100 to 500. In the graphs, "One-Hop" refers to the One-Hop Broadcast scheme, and "802.11 TSF" refers to the IEEE 802.11 TSF applied to multi-hop networks. These schemes were explained and discussed in section 4.2. "MTSP-m" refers to our proposed protocol, with the suppress threshold m. So in "MTSP-2" scheme, a leaf node suppresses its beacon if it receives two other beacons in the beacon interval. Figure 7(a) shows the maximum synchronization error over the whole simulation time. Since all nodes transmit beacons at every beacon interval in One-Hop Broadcast scheme, it achieves the best accuracy among all protocols. However, it is achieved at a very high cost, as shown in Figure 7(b).

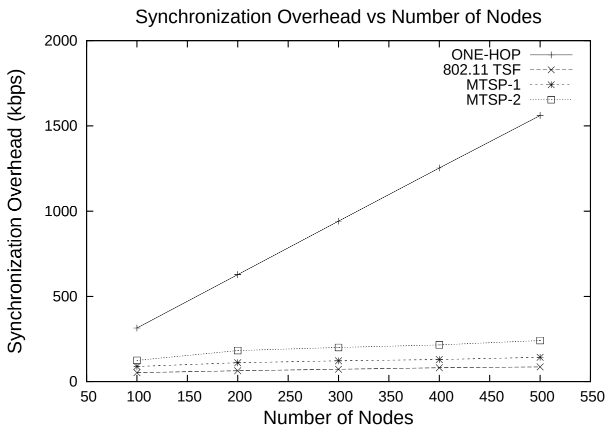
On the other hand, 802.11 TSF protocol introduces the least amount of overhead. This is because only one node transmits beacon in a broadcast region. Thus, the amount of overhead in 802.11 TSF stays constant as the node density increases. However, the accuracy of 802.11 TSF is much worse than other two protocols.

MTSP-1 and MTSP-2 achieves accuracy comparable to One-Hop Broadcast, at a cost much less than the cost of One-Hop Broadcast. As we can see in 7(b), the overhead of MTSP grows slowly with increasing number of nodes. In terms of accuracy, MTSP-2 achieves a slightly better accuracy than MTSP-1. This is because each node has more chance of receiving beacons from faster nodes at each beacon interval. However, MTSP-2 pays approximately 80% more overhead over MTSP-1.

Figure 8(a) and 8(b) show the impact of suppress threshold on the accuracy and overhead of MTSP. The simulation is done in 1000m $\times$ 1000m area with 200 nodes, and the suppress threshold m is varied from 2 to 7. The result shows that MTSP with higher suppress threshold achieves better accuracy in average, at a cost of additional communication overhead. However, the benefit is not significant compared to the additional overhead. So it is sufficient to just use 1 or 2 as the suppress threshold.

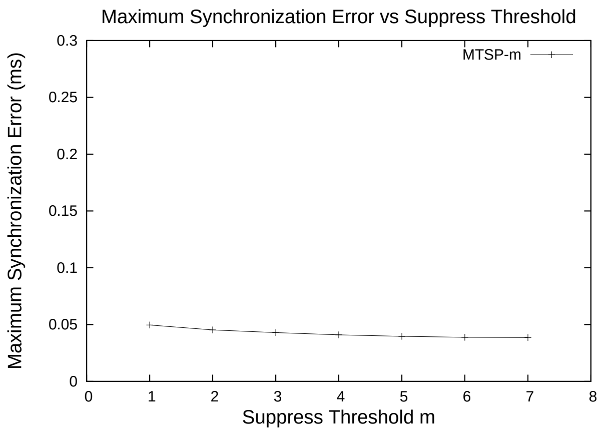


(a) Maximum synchronization error

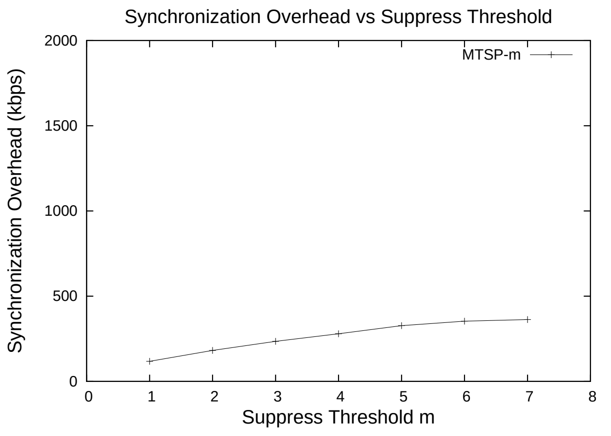


(b) Synchronization overhead

Figure 7: Maximum synchronization error and Synchronization overhead vs. number of nodes.



(a) Maximum synchronization error



(b) Synchronization overhead

Figure 8: Maximum synchronization error and Synchronization overhead vs. suppress threshold.

Now we study the impact of controlling transmission power on MTSP. When a higher transmission power is used, the transmission range increases and thus the depth of the synchronization tree will be smaller. So the network synchronization error is expected to be lower.

For this simulation, we have used a 500m $\times$ 500m area with 200 nodes. Figure 9(a) and 9(b) shows the maximum synchronization error and the overhead, when using different transmission range.

For MTSP, the results show that if energy consumption is not an issue, we can benefit from using higher transmission power. Both the synchronization error and the overhead goes down as the transmission range of a node increases. The reason that the overhead of MTSP is reduced when using higher transmission power is because the number of leaf nodes in the synchronization tree decreses when higher power is used.

6. CONCLUSION

In this paper, we have proposed MTSP, a time synchronization protocol for multi-hop wireless networks. MTSP is designed to work under the assumption that the synchronization process must also be synchronized. This requirement is valid for many applications such as power management schemes and multi-channel MAC protocols that require time synchronization among nodes.

Since a node only synchronizes to a faster node, the synchronization protocol must make sure that every node is synchronized to the fastest node. MTSP achieves this by implicitly building up a synchronization tree rooted at the fastest node in the network. Thus, when the network is static, the network eventually converges to a steady state where an upper bound on network synchronization error is guaranteed, assuming that there is no beacon packet loss. This upper bound information can be used to set up a time margin before starting each new beacon interval. If the topology changes, the network self-stabilizes to a steady state after the network becomes static again. We have shown that the protocol eventually converges to the steady state, and also calculated the upper bound on network synchronization error in the steady state.

Also, using the tree structure, MTSP 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 or using up a large amount of energy.

The simulation results show that IEEE 802.11 TSF has the least amount of overhead. However, its accuracy is very low. On the other hand, the One-Hop Broadcast scheme where every node transmits beacon at every beacon interval shows the best performance in terms of accuracy, but the overhead is very high. MTSP is shown to achieve a reasonable accuracy at the cost of low overhead.

We also have studied the impact of transmission power using MTSP. When nodes use higher transmission power, the accuracy increases because the depth of the synchronization tree is reduced. For MTSP, the communication overhead decreases as the transmission power increases, because the number of leaf nodes in the synchronization tree increases. So MTSP can benefit from using higher power for synchronizaiton, if energy consumption is not an issue.

If we take energy into account, the trade off between energy consumption and the accuracy of synchronization would make an interesting problem. If higher power is used, more energy is consumed, but a higher accuracy is achieved using less communication overhead. So we can reduce the synchronization frequency to reduce energy consumption, while still achieving the desired level of accuracy. So given a network topology, one might be able to find the optimal tuning of transmission power and synchronization frequency, so that a desired level of accuracy is achieved consuming the least amount of energy. We are planning to extend our research in this direction in the future.

7. REFERENCES

[1] B. Chen, K. Jamieson, H. Balakrishnan, and R. Morris. Span: An energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks. In *Mobile Computing and Networking*, pages 85–96, 2001.

[2] D. Mills. Internet Time Synchronization: The Network Time Protocol. *IEEE Trans. Communications.*, October 1991.

[3] IEEE 802.11 Working Group. Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications, 1997.

[4] J. Elson and D. Estrin. Time Synchronization for Wireless Sensor Networks. In *International Parallel and Distributed Processing Symposium (IPDPS)*, April 2001.

[5] 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.

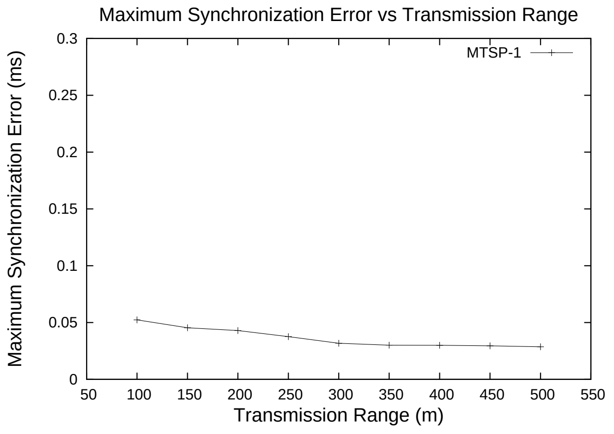
[6] Jana van Greunen and Jan Rabaey. Lightweight Time Synchronization for Sensor Networks. In *Workshop on Wireless Sensor Networks and Applications (WSNA)*, September 2003.

[7] Kay Römer. Time Synchronization in Ad Hoc Networks. In *ACM MOBIHOC*, October 2001.

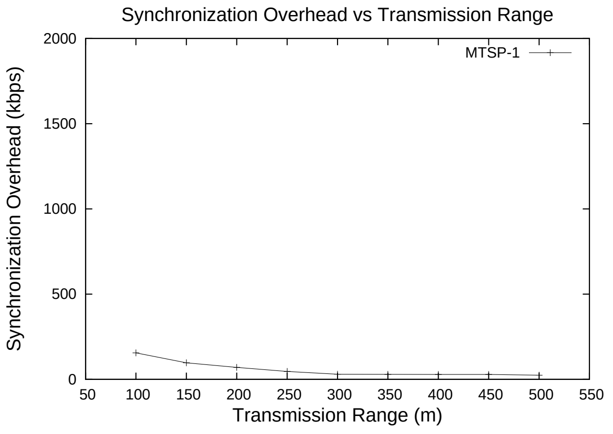
[8] L. Huang and T. Lai. On the Scalability of IEEE 802.11 Ad Hoc Networks. In *ACM MOBIHOC*, June 2002.

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

[10] 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.



(a) Maximum synchronization error



(b) Synchronization overhead

Figure 9: Maximum synchronization error and Synchronization overhead vs. transmission range.

[11] S. Ni, Y.C. Tseng, Y.S. Chen and J.P. Sheu. The Broadcast Storm Problem in a Mobile Ad Hoc Network. In *ACM MobiCom*, 1999.

[12] The CMU Monarch Project. Wireless and Mobility Extension to ns.

[13] VINT Group. UCB/LBNL/VINT network simulator ns (version 2).

[14] 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.

APPENDIX

A. PROOF OF SELF-STABILIZATION

In this section, we prove Theorem 1. We prove this by saying that if a node always chooses the fastest neighbor as its parent, it will eventually point towards the fastest node in the network, meaning that the node will join the synchronization tree rooted at the fastest node. So if every node always chooses its parent this way, all nodes will eventually point towards the root, and the synchronization tree rooted at the fastest node will be established. Once the synchronization tree is established, it stays unchanged if the network is static, because a node does not switch parents. Thus, the protocol enters a steady state.

First, we start with a simple example, and generalize the argument to any possible cases.

Consider the scenario in Figure 10. 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 regards A as the root of the synchronization tree, which is the fastest node in the network. On the other hand, D regards E as the root. 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.

The requirement implied in the above argument is that node C receives beacons from B and D every once in while so that C can eventually point toward the root. It may be the situation that B and D are already identified as leaf each in a different tree. Then, B and D do not transmit beacons in every beacon interval. However, as discussed in section 4.3, due to the requirement that forces leaf nodes to transmit beacons at least after r iterations of silence, the requirement is met.

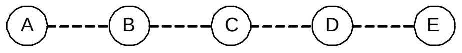


Figure 10: A network scenario with 5 nodes.

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 the analysis here, we ignore the impact of ϵ_{max} in Equation 12. 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,

$$\Delta_{AB} = (\alpha_A - \alpha_B)L \tag{15}$$

Ignoring nodes D and E for now, there can be two situations at the kth beacon interval: Either node A or B can transmit a beacon first. If node A sends the beacon first, node B will synchronize to A and then transmit its beacon. In this case,

$$\Delta_{BC} = T_B - T_C = T_A - T_C = \alpha_A t + \delta_A - (\alpha_C t + \delta_C) \tag{16}$$

If node B transmits a beacon before node A, then Δ_{BC} would be:

$$\Delta_{BC} = T_B - T_C = T_A - \Delta_{AB} - T_C = \alpha_A t + \delta_A - (\alpha_A - \alpha_B)L - (\alpha_C t + \delta_C) \quad (17)$$

In both situations we obtain:

$$\Delta_{BC} = (\alpha_A - \alpha_C)t + u \quad (18)$$

where u is a constant (In the former case, $u = \delta_A - \delta_C$, and in the latter case, $u = \delta_A - (\alpha_A - \alpha_B)L - \delta_C$). Similarly,

$$\Delta_{CD} = (\alpha_E - \alpha_C)t + v \quad (19)$$

where v is a constant. Since $\alpha_A > \alpha_E$, 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 if the network is static, every node in the network will eventually choose its parent towards the fastest node, which means every node will eventually join the synchronization tree rooted at the fastest node.

We prove this using the previous argument and 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.

$$\Delta_{A_1 R} \geq \Delta_{A_1 i} \quad (20)$$

Now suppose node A_{k+1} is $k+1$ hops away from node R. It has a neighbor node A_k, which is already in the synchronization tree rooted at R. All of A_k's ancestors, A_1, A_2, ..., A_{k-1}, are also in the synchronization tree rooted at R.

Since the nodes A_i $(i = 1, 2, ..., k)$ already in the synchronization tree rooted at R, from Equation 30 in Appendix B, beacon interval, the maximum of $\Delta A_k R$ at the start of a beacon interval is:

$$\max \Delta_{A_k R} = \sum (\alpha_R - \alpha_i)L \quad (21)$$

where $i \in P_{A_k R}$. $P_{A_k R}$ is the set of nodes that are in the path from node A_{k+1} to node R. Thus,

$$\min \Delta_{A_{k+1} A_k} = T_R - (\sum \alpha_R - \alpha_i L) - T_{A_{k+1}} \quad (22)$$

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

$$\Delta_{A_{k+1} A_k} = (\alpha_R - \alpha_{A_{k+1}})t + c \quad (23)$$

where c is a constant. Similarly, if another neighbor of node A_{k+1}, B, has node S as its root, then

$$\Delta_{A_{k+1} B} = (\alpha_S - \alpha_{A_{k+1}})t + c' \quad (24)$$

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, meaning that the every node joins the synchronization tree rooted at the fastest node. Once the tree is established, the parent of each node does not change given that the network stays static. Thus the network will eventually enter the steady state, where a synchronization tree rooted at the fastest node is established.

B. UPPER BOUND ON THE SYNCHRONIZATION ERROR

Given that the network is in the steady state, we can calculate the upper bound on the clock difference between a node and the fastest node in the network. 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 synchronization tree. 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} (in Equation 12) in the analysis. The effect of ϵ_t is added to the upper bound at the end of the analysis.

To make the worst case, we assume that the order of transmission in each beacon interval is C_k, C_{k-1}, ..., C_1, R.

At ith beacon interval, node R sends a beacon to node C_1, so 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:

$$T^{i+1}_{C_1} = T^i_R + \alpha_{C_1} L \quad (25)$$

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:

$$T^{i+2}_{C_2} = T^{i+1}_{C_1} + \alpha_{C_2} L = T^i_R + (\alpha_{C_1} + \alpha_{C_2})L \quad (26)$$

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:

$$T_{C_k}^{i+k} = T_R^i + (\sum \alpha_{C_j})L \tag{27}$$

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

$$T_R^{i+k} = T_R^i + \alpha_R kL \tag{28}$$

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

$$\Delta_{C_k R} = T_R^{i+k} - T_{C_k}^{i+k} = \alpha_R kL - (\sum \alpha_{C_j})L = (\sum (\alpha_R - \alpha_{C_j}))L \tag{29}$$

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

Suppose D is the network diameter, which is the maximum hop distance between any pair of nodes in the network. Then the maximum clock difference between any pair of nodes in the network will be:

$$\max \Delta = (\sum (\alpha_R - \alpha_{C_j}))L \tag{30}$$

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

Since the clock rates are unknown, a node cannot precisely determine $\max \Delta$. So we can just assume the worst case, where R has the maximum allowable 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:

$$\max \Delta = (\sum (\alpha_R - \alpha_{C_j}))L = 2fDL \tag{31}$$

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:

$$\max \Delta = 2fDL + D\epsilon_{max} \tag{32}$$

If we assume f is 0.0001, D is 10, L is 100ms, and ϵ_{max} is 1μs, then the maximum network synchronization error will be 210 μs.

This is a very conservative calculation of the maximum synchronization error given the clock accuracy requirement, because it is obtained from considering all possible worst cases. In reality, the clock rates are not distributed as we assumed, the maximum hop distance from the fastest node may be less than D, and the order of synchronization may vary all the time. So the maximum network synchronization error of MTSP is expected to be lower than what we have calculated.