Project Report

Sayed Ahmed and Dinesh Bhat

May 16, 2004

Course:	74.783 Mobile Networking
Project:	Time Synchronization in Sensor Networks Using Flooding
Email:	sayed@cs.umanitoba.ca & bhatdinu@cs.umanitoba.ca
Instructor:	Dr. Jelena Misic

Abstract

One of the most important issues in distributed systems is time synchronization. Many distributed applications that collects data require that the distributed objects must be able to synchronize their time with a single reference point and time stamp the data in order to perform global transactions with high precision and consistency. Time synchronization is also needed to perform tasks to provide coordination and data consistency. Time synchronization protocols for sensor networks must be suitable to the resource and power limitations in sensor nodes. The existing protocols for other kind of networks need to be extended and modified to fit the requirements of wireless sensor networks. This project implements flooding time synchronization protocol(FTSP) on a set of sensor motes running TinyOS operating system. The algorithm provides high precision, negligible synchronization error, robustness against node failures and topology changes for a small to medium sized sensor networks. Moreover, the algorithm uses less communication bandwidth to provide power efficiency. In this report, we presented our implementation of the FTSP protocol. We extensively studied and tested the implementation to analyze different issues like initial synchronization time, root election time, clock drifts, signal strength, scalability, effect to topology changes and node failures for varying network structures using Mica2 and Mica2Dot motes. An in depth analysis of our experiment is also provided in the report.

1 Introduction

Technological Advances in Electronics have led to highly efficient, low powered, integrated communication devices and sensors. Sensors can be spread throughout a region to build a network for many applications such as environment observation, habitat monitoring, protecting a region from intruders, military applications and so on. Usually a wireless sensor network consists of many sensors (clients) and a basestation (usually a sensor). The client sensors are spread over a region and the base station is usually connected to a workstation(PC) where the sensor network operating system runs. An application operating in the workstation maintains and collects data from the sensor network. Client sensors also run some application to send data to the workstation through the base station for offline querying and processing. Recently, sensor networks have become a very active topic of research due to its emerging importance in many personal, home, industry, agriculture, and medical applications. In many applications, the sent data needs to be time stamped by the sensors to be useful. Hence, time synchronization is important in sensor networks. However, as sensors lack in resource, power, and processing capability, the time synchronization protocols must be power efficient.

In the project, we implemented the Flooding Time Synchronization Protocol which is effective for both wide area multihop distributed sensor networks as well as for small area single hop networks. The protocol uses MAC layer time stamping and compensates for the skew compensation due to the clock drifts (small difference in clocks) of the sensor nodes using linear regression. Broadcast based synchronization and the skew compensation reduces communication overhead helping in power conservation.

Our target platform [**?**] consists of Mica2 and Mica2Dot motes. Mica2 has 7.37 MHz processor, 4KB of RAM, flash memory of 128KB, wireless transceiver of 916 MHz. It also comes with sensors that sense temperature, humidity, and light. Mica2Dot has 10-bit analog to digital converter, 916 MHz of base radio frequence and it also comes with sensor boards to capture light, humidity and so on. Mica2 has 921.6 KHz timer where Mica2dot has 500 KHz timer. nesC language is used to program the motes. TinyOS [?] is an "open source, event driven modular" operating system. TinyOS handles the tasks of typical operating system like concurrency, scheduling, radio communications, clocks, timers, power management, resource allocation, ADC, I/O, EEPROM and so on. TinyOS can be used with the sensor networks to perform operations that are associated with the sensors.

The protocol is implemented on Mica2/Mica2-Mica2dot platforms running TinyOS. In all cases, we found the protocol is well suited for sensor networks providing precision in the microsecond range and negligible synchronization error. The rest of the report is organized as follows: In section 2, we describe the algorithm, the experimental test-bed in section 3, in section 4, analysis and evaluation and finally, we conclude with conclusions and future works.

2 Related Works

Many time synchronization protocols such as Network Time Protocol(NTP) [4], Reference Broadcast Synchronization(RBS) [2], and Timing-sync Protocol for Sensor Networks(TPSN) [5] have been developed. NTP uses GPS to synchronize to the NTP time servers. NTP is the most prominent in Internet domain and provides precision in millisecond range. Non-determinism in wireless sensor networks(WSN) makes NTP suitable only for low precision WSN applications. RBS uses reference broadcast and recorded time interchange with broadcaster. However, RBS has not been extended for multihop WSN. The TPSN uses a hierarchical structure and pair-wise synchronization along the edges. TPSN does not support clock drifts and dynamic topology changes.

3 FTSP Algorithm

The FTSP protocol [3] provides global time synchronization in the network. All of the nodes in the network including root, run the same FTSP algorithm. The nodes selects a node among them as the root and synchronize the global time of each node to the local time of the root. Root is usually the node having the minimum node id. To make each node synchronized, each node periodically broadcasts Time Synchronization messages containing estimate of global time and the local time once in every x(predefined) seconds. Each node keeps a certain number of past messages (a new synchronization point) in a table as local time and offset (globaltime-localtime) pair. Each node uses this table to (re)calculate the skew and offset of his local clock vs global time when a new TimeSync message comes. The local time of node A is assumed to be approximately a linear function of local time of node B. So the algorithm uses linear regression. This skew is needed to compensate for clock drifts. The equation a client uses to calculate global time is as follows

globalTime = localTime + offset + skew * (localTime - syncPoint)

In case of single hop, all broadcasted message are heard by everyone. However, in case of multihop only the neighbors can hear the time synchronization messages. The global time (i.e. local time of the root) from root propagates to the farthest part using neighborhood broadcast. Therefore in multihop environment root selection time is greater. Also, the synchronization error is little bit greater.

4 Our Implementation of the FTSP Algorithm

We used the built-in SysTimeStamping component of the TinyOS to time stamp incoming and outgoing messages using MAC layer time stamping. The broadcasted Time Synchronization message contains the node ID of the current synchronization root, the node ID of the sender, a sequence number that is only incremented by the root, and the global time of the sender at transmission time.

4.1 Phases of algorithm:

- INIT: This phase is triggered when we just turn on a mote. The mote first tries to listen time synchronization messages from the network for certain amount of time. If it does not receive any such message from the network it declares itself to be the root and embeds its node-id as the root in outgoing messages.
- SENDING: This phase is triggered after a fixed time interval. We use TinyOS TimerC component to provide timer fire event. At this event, each node broadcasts time synchronization messages embedding current root id. Because of this, ultimately the node with the smallest node ID is selected as the root of the network. The root is the only node who increases the sequence number of the time synchronization message. However, at the instant of finding a new node with smaller id we do not force all the network to change its global time by a large amount rather we wait for a certain period of time (ROOT_SWITCH_EVENT_COUNT), during which the new root determines the skew and the offset of the global time and only then announces itself to be the root. Skew and offset values do not change if root does not change. We keep the skew value to be 0 to decrease the

error in arithmetic. A node also declares itself the root if he did not hear any time synchronization messages for ROOT_SWITCH_EVENT_COUNT Timer events.

- PROCESSING: This phase is triggered when a time synchronization message is received. First, we check for the root id of the incoming message and if the root id is smaller than our root id after waiting for ROOT_IGNORE_EVENT_COUNTS timer events our root id is changed. Besides, we store the offset and localtime pair in our data table if we get a new synchronization point and calculate the skew and offset of our local time with respect to the global time. The message is considered to be new only if it has higher sequence number than the last heard message. The skew is normalized to zero by subtracting one from the slope of the best fitting line to provide greater precision. Mentionable, in our experiment, we simulated multihop routing where a node would process incoming message only if the message is from the neighbors otherwise it would just drop the message. Hence, clients can be placed close to each other or base station even in multihop.

5 Experimental Testbed

The experimental testbed is based on single hop and multihop routing strategies. We implemented and tested the algorithm for both single hop and multihop setup. Moreover, we used homogeneous environment containing only mica2 motes and heterogeneous environment containing both mica2 and mica2dot motes for both single hop and multihop setup.

- Single Hop Routing

  In case of single hop setup, all broadcasted messages from all nodes are heard by

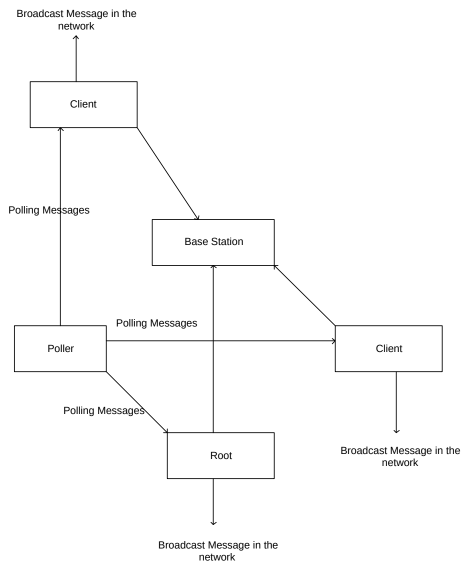
Broadcast Message in the network
Client
Polling Messages
Base Station
Polling Messages
Poller
Client
Polling Messages
Root
Broadcast Message in the network
Broadcast Message in the network

Figure 1: Single Hop Network

everyone. All of them are within the radio range of the base station. Nodes hear data from all other nodes and synchronize their time with the currently selected root. Parallelly, root election algorithm works and consequently the node with the smallest id becomes the root. The figure 1 shows single hop routing network. In single hop routing, the nodes need to be within the radio range of the base station. However, the nodes send the messages to the base station directly.

- Multihop Routing

  In case of multihop setup only the neighbors of a node can hear the time synchronization messages from it. The time synchronization message from root propagates to the farthest part using neighborhood broadcast. Therefore, in multihop environment root selection time is greater than single hop. Also, the synchronization error is little bit greater than single hop. In multihop architecture, the nodes can be out of range of the base station. Message routing takes place by sending the message to the neighboring nodes. If the clients or the root is not within

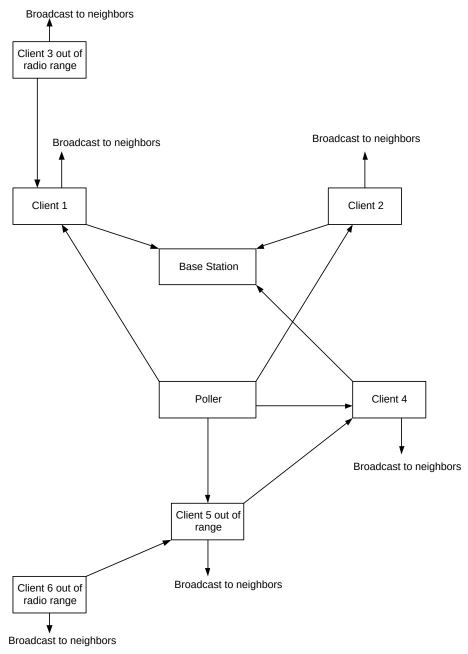

Figure 2: Multihop Network

the radio range of the base station they use some nodes as intermediate nodes to deliver their data to the base station. The figure 2 shows multihop routing network.

5.1 Based on Hardware Platform

- Homogeneous Network:

  Homogenous environment consists of client nodes of same architecture. We have tested our implementation with keeping Mica2s.

- Heterogeneous Network

  Heterogeneous environment consists of client nodes of different architectures. In heterogeneous setup, we used both Mica2 and Mica2dots as clients. Both types of motes interacted with each other to transmit the time synchronization messages among the nodes to make their time synchronized with the root. broadcaster.

Parameters to study

- Initial Time for Network Synchronization

  At the network start up, initially it takes some time to select the proper root and also to get all the clients synchronized with the root. For FTSP protocol, this time is much less.

- Average and Maximum Synchronization error

  To calculate average synchronization error, we first calculate the average of the global times reported by all the clients at a particular time instant. Then for each node the difference of their global time from this average is calculated. To get average error, the average of these differences is calculated. To get max error, the maximum of these differences is calculated. These are the average and maximum synchronization error at that time instant. We calculated the average of all the average and maximum synchronization error of every time instant to get the global average and maximum synchronization error and we found the errors in microsecond range.

- Polling Interval

  The effectiveness of a time synchronization protocol also depends on the interval time at which synchronization messages are broadcasted to the network. If a protocol provides synchronization with reduced error when the interval is much longer the protocol is a good one which can also be used to use less power.

- Arbitrary Client Failure

  Client failure has no or minimal effect in time synchronization.

- Root Failure

The root can also die due to insufficient power. If the already elected root dies in the network then the leader election protocol must reelect a new root and compute the global time and broadcast it to all the nodes to synchronize the entire network. FTSP requires a very short interval to reelect the root and make the network synchronized.

6 Analysis

A detailed analysis of our experiment is provided in this section. In homogeneous environment, we used two Mica2 motes as the clients. The beacon broadcaster and the base station were also Mica2 motes. In case of heterogeneous environment, we used two Mica2s and three Mica2dot motes as clients. Again the reference broadcaster and the base station were Mica2 motes. We programmed all the motes and turned on all the motes nearly at the same time. After the network is synchronized with the root we waited for some time (15-30 minutes) then turned off the root to observe the effects on synchronization. Similar procedure was followed for all experiments.

6.1 Homogeneous Single Hop Environment

Figure 3 shows the average synchronization error before the synchronization point is reached and the figure 4 shows the average synchronization error after synchronization point for homogeneous single hop environment. Mentionable as we have only two client motes average and maximum synchronization errors are all the same. All the nodes were turned on almost at the same time. At the startup, since there is no root selected, all nodes announce themselves as the root. After 3.5 minutes, the root (smallest node-id) is properly selected and all clients are able to identify the root. However, at this moment all clients are not synchronized with the root. After 5.5 minutes, all nodes are synchronized with the root. Synchronization error is the highest in this period (until synchronization point is reached). Once, the root is decided, all nodes synchronize with the root. Then we left the network to run for 30 minutes. During this time, the average synchronization error is 0.56 micro sec where the maximum error reached to 2 micro sec in a very rare case. Then we turned off the root. For some time the clients were reporting their previous root as the root. But after a short while (2.5 minutes), again another node (smallest node-id) was properly selected as the root. As we have only one node at this point the synchronization error is zero because the node is always synchronized with itself.

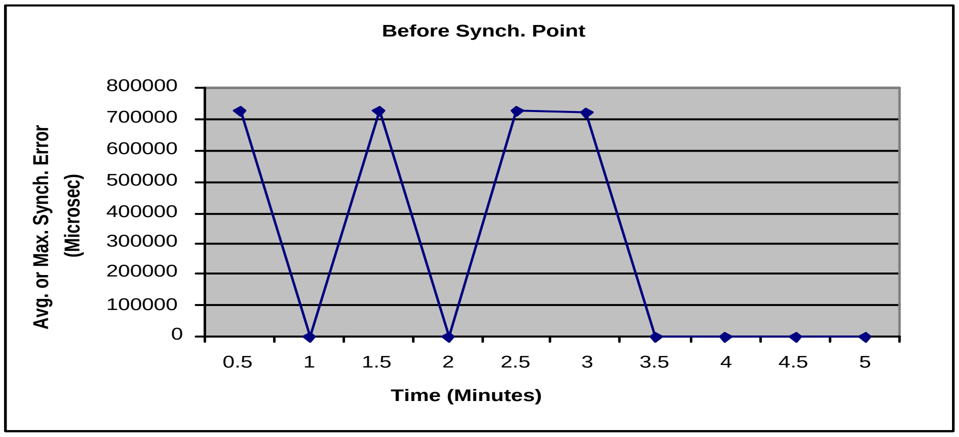

Figure 3: Before reaching the network synchronization

6.2 Homogeneous Multihop Environment

Figure 5 shows the average synchronization error before the synchronization point is reached and Figure 6 shows the average synchronization error after the synchronization point. Also, in this setup initially all nodes declare themselves to be the root but

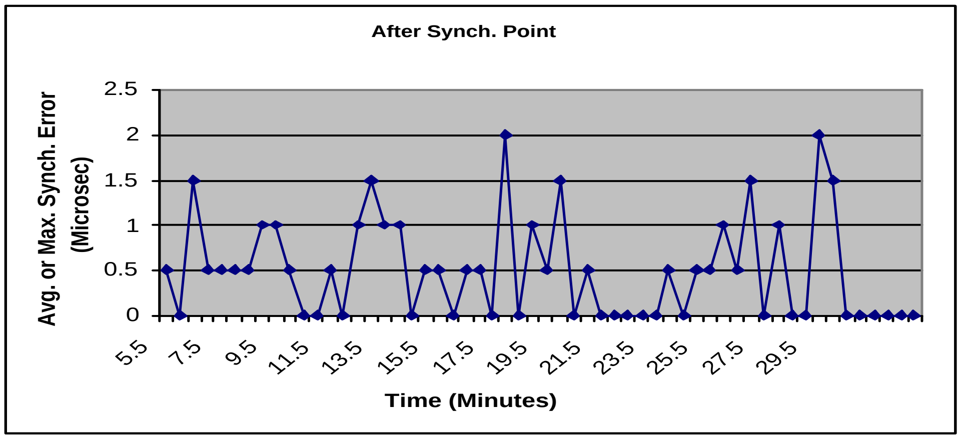

Figure 4: After reaching the network synchronization

ultimately at 4 minutes the root is properly selected. As time synchronization messages pass only through neighbors, root is selected 30 seconds later than single hop. Network was synchronized at 6 minutes. We ran this experiment for another 30 minutes before turning the root off. The average synchronization error in this case is 0.37 micro sec where the maximum error reached to 1.5 micro sec in a very rare case. Then we turned off the root. For some time the clients were reporting their previous root as the root. But after a short while 2.5 minute again another node was properly selected as the root. As we have only one node, now the sync error in our case is zero because the node is always synchronized with itself.

6.3 Heterogeneous Single Hop Environment

Figure 7 shows average and maximum synchronization error before synchronization point is reached and figure 8 shows average and maximum synchronization error after synchronization point is reached for heterogeneous single hop environment. At the startup, everybody declares itself to be the root but after 5 minutes the root is properly selected. This selection time is larger than homogeneous environment as the Mica2

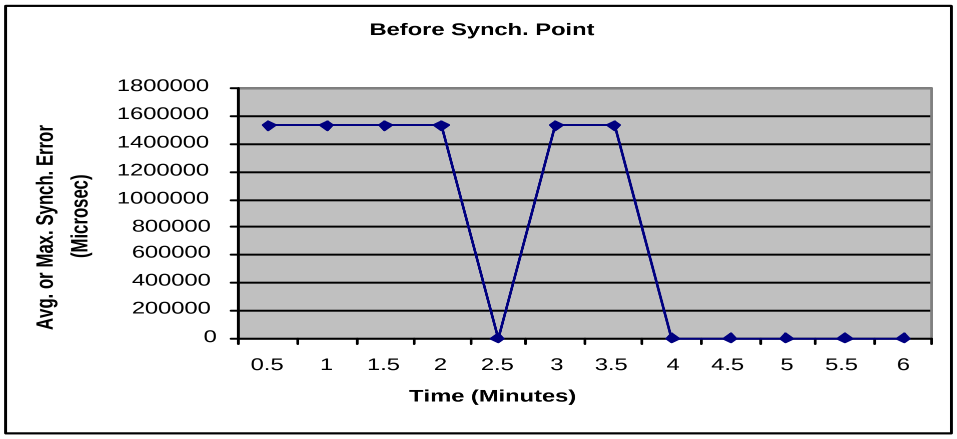

Figure 5: Before reaching the network synchronization

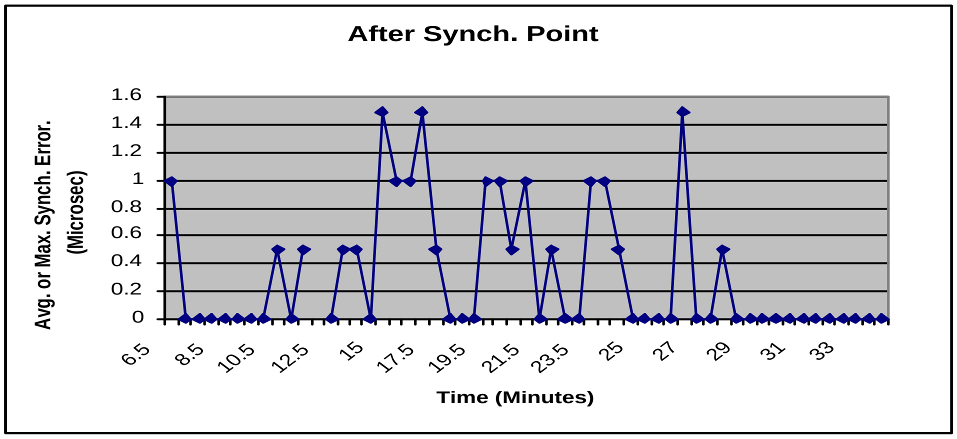

Figure 6: After reaching the network synchronization

and Mica2dot have clock difference between them. Moreover, other architectural differences, processing capability, packet loss can cause this larger initial synchronization time. However, yet the network is not synchronized with the root. At 6.5 minutes we got every node synchronized with the root. Also this time is larger than homogeneous environment because of the causes mentioned above. Note able synchronization error is the maximum until synchronized point is reached. Once, the network is synchronized, synchronization error reaches minimum. We kept the network in this state for a while and the average and maximum synchronization error are 3.37 and 6.73 micro sec respectively. These errors are also higher than homogeneous environment due to heterogeneous motes.

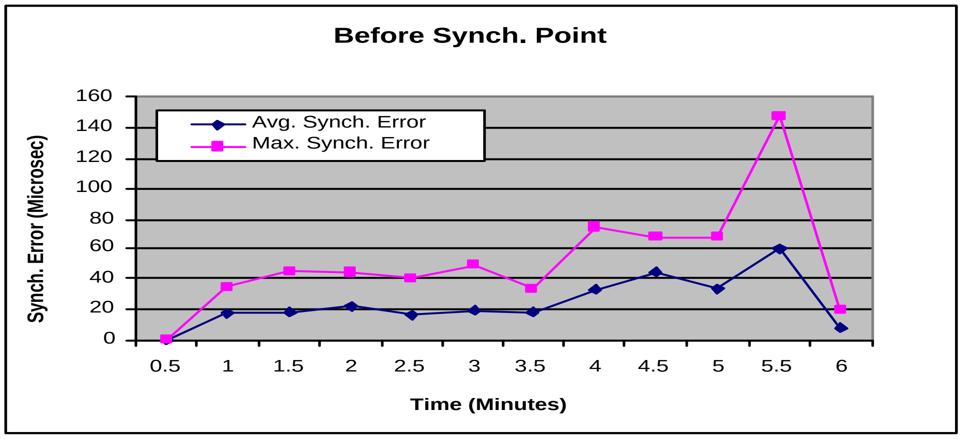

Figure 7: Before reaching the network synchronization

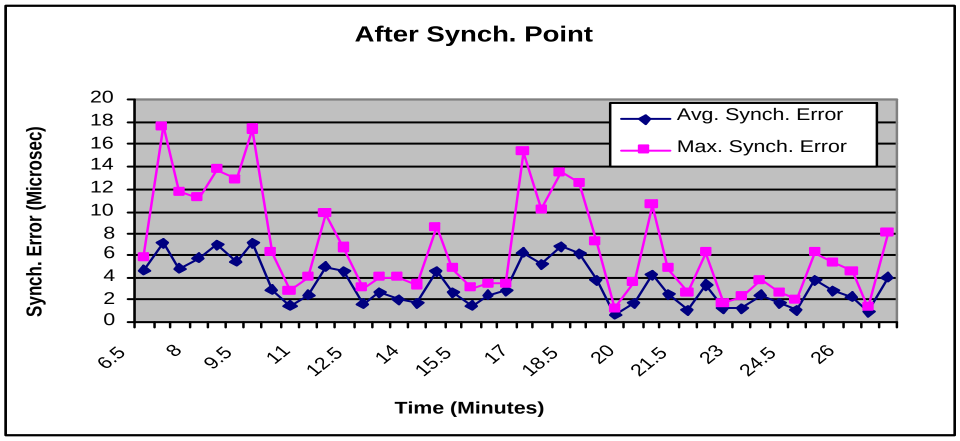

Figure 8: After reaching the network synchronization

6.4 Heterogeneous Multihop Environment

Figure 9 shows average and maximum synchronization error before synchronization point is reached and figure 10 shows average and maximum synchronization error after synchronization point is reached for heterogeneous multihop environment. At the start of the network everybody declares itself to be the root as no root is selected yet. After 6 minutes the root is properly selected. This selection time is larger than single hop heterogeneous environment as time synchronization messages only traverse through neighbors it takes more time to go through all nodes. However, yet the network is not synchronized with the root. At 20 minutes we got every node to be synchronized with the root. This synchronization time is larger than homogeneous environment also larger than heterogeneous single hop environment. Because of multihop environment the local time of root propagates only through the neighborhood broadcast taking long time to reach to the farthest part. Besides, now the environment is heterogeneous. Hence, it takes too long to get the network synchronized. In this period (until sync. point), as usually the synchronization error is maximum. We kept the network in this synchronization state until 31 minute. The average and maximum synchronization error in this synchronization period are 3.44 and 5.37 respectively. The average error is slightly more than single hop, but the max error is a little bit higher. After synchronization as the network is stable, all nodes get data from their neighbors properly, synchronization error does not differ much with the single hop scenario.

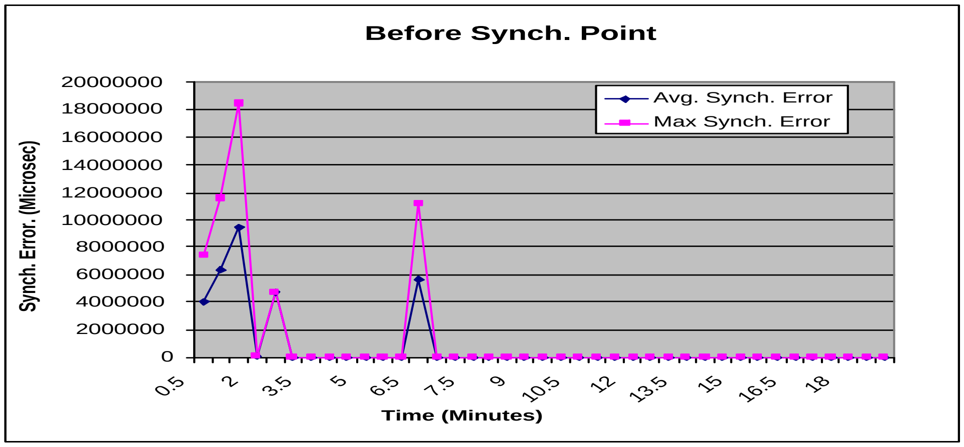

Figure 9: Before reaching the network synchronization

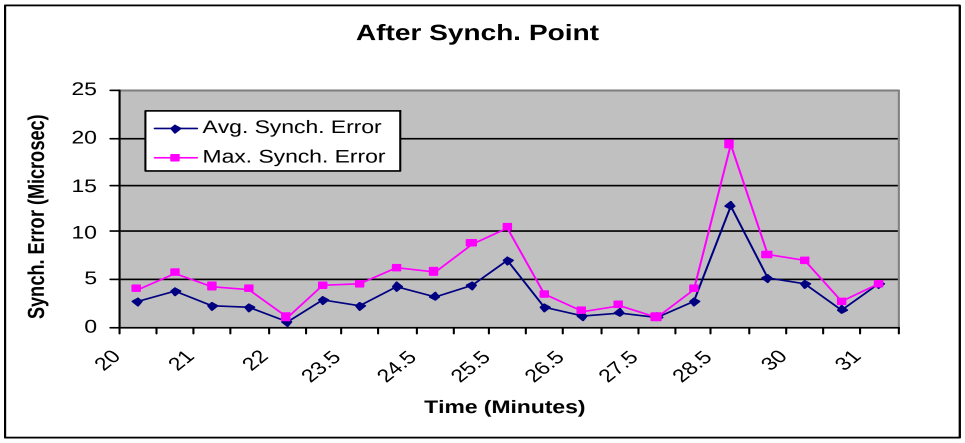

Figure 10: After reaching the network synchronization

7 Conclusions and Future Works

We have implemented the Flooding Time Synchronization Protocol for both single hop and multihop wireless sensor networks. The implementation was done both for homogeneous environment containing only Mica2 motes and heterogeneous environment containing both Mica2 and Mica2Dot motes running TinyOS operating system. We found a very short initial synchronization time. Also, the implementation was robust to the root failure and client failure. For homogeneous environment the average synchronization error is negligible. The error is 0.56 micro sec for single hop and 0.37 micro sec for multihop. We tested our implementation for a heterogeneous environment where the error was 3.37 micro sec for single hop and 3.44 micro sec for multihop. Mentionable, to our knowledge no experimental result exist for a heterogeneous FTSP environment. Our implementation provides results for the heterogeneous environment. Our future work may focus on the testing of the implementation for a wide sensor network consisting large number of sensors. Besides, the TSync [1] algorithm which was implemented for sensors containing GPS can be implemented for Mica2 and Mica2dot environment with some modifications.

References

[1] Crossbow. Wireless sensor network kit. http://www.xbow.com.

[2] R. Han H. Dai. Tsync : A lightweight bidirectional time synchronization service for wireless sensor networks. *ACM SIGMOBILE Mobile Computing and Communications Review, Special Issue on Wireless PAN and Sensor Networks*, 8(1):125–139, Jan 2004.

[3] L. Girod J. Elson and D. Estrin. Fine-grained network time synchronization using reference broadcasts. *Proceedings of the fifth symposium OSDI*, Dec 02.

[4] G. Simon M. Maroti, B. Kusy and A. Ledeczi. The flooding time synchronization protocol. *TECHNICAL REPORT*, 2004.

[5] D. L. Mills. Internet time synchronization: The network time protocol. *IEEE Computer Society Press*, 1994.

[6] University of Berkeley. Tinyos operating system. http://webs.cs.berkeley.edu/tos/.

[7] M. B. Srivastava S. Ganeriwal, R. Kumar. Timing-sync protocol for sensor networks. *SenSys*, Nov 2003.