# Time Synchronization Algorithms based on Timing-Sync Protocol in Wireless Sensor Networks

Ali Burak Kulakli
Computer Engineering Dept.
Izmir Institute of Technology
Urla, Izmir 35345, Turkey
Email: alikulakli@iyte.edu.tr

Kayhan Erciyes
International Computer Institute
Ege University
Bornova, Izmir 35100, Turkey
Email: kayhan.erciyes@ege.edu.tr

***Abstract*—Wireless Sensor Networks(WSN) are large scale networks of sensors running on wireless environment. For an application running on a WSN, gathered data by the sensors are time critical in most of the cases. However, almost all the nodes suffer from a problem named clock drift. This problem causes clock difference among nodes as time goes because the processors do not run exactly at the same speed. There are many proposed solutions to remedy this problem. TPSN (Timing-sync Protocol for Sensor Networks) is one of the effective protocols proposed to synchronize sensor networks. In this paper, we propose enhancements over TPSN to synchronize nodes in a wireless sensor network more effectively with a lower message complexity and higher precision.**

***Index Terms*—Wireless sensor networks, clustering, spanning tree, time synchronization, TPSN**

## I. Introduction

Time synchronization on WSN (wireless sensor networks) is a well-known problem. Most of the applications need time critical data values from the nodes in the network. However, clocks of the individual nodes may differ at the same reference time after a while if no action is made to synchronize them. That will possibly cause application errors eventually because the input data is incorrectly timestamped and they will not be interpreted correctly by the application. For the time critical applications running on WSN, clock drift problem should be reduced to a reasonable level or completely eliminated if possible.

Even a network that consists of two nodes may have two clocks labeled with same frequencies, these clocks will not be running at exactly the same speed due to various reasons. This tiny difference cumulates as time goes and after a while if there is no interference to synchronize the nodes, clocks will have significant differences. In addition to clock drift, some nodes might be started later then the other nodes in the network or some nodes can be added to the network later. In such cases, there would be a fixed clock difference offset. This is also another problem which should be solved. [1]

Sensor nodes are very tiny instruments and running with a limited energy so it is not easy to synchronize nodes effectively because of energy consumption. Also, they usually have not much processing power and this prevents any complex algorithms to run on WSN. [2]

There are several time synchronization protocols due to varying requirements, such as precision or degree of mobility. Time synchronization procedure typically is a message exchange containing the timestamp and the measurement of delay. There are three basic solutions for time synchronization in sensor networks [4]:

1) Sender-Receiver Based Synchronization
2) Receiver-Receiver Based Synchronization
3) Delay Measurement Synchronization

Receiver-Receiver based synchronization algorithms commonly use one-way message exchange such as in the Reference Broadcast Synchronization (RBS) [5]. On the other hand, two-way message exchange is used in Sender-Receiver based synchronization protocols, such as the Timing-sync Protocol for Sensor Networks (TPSN) [6]. There are also some synchronization protocols based on one-way message exchange as well as the measurement of delay. An example of such a protocol is Delay Measurement Time Synchronization (DMTS) [7].

TPSN is easy to implement and an effective time synchronization protocol but it has some weaknesses. When number of levels increases, local clock offset difference also increases. Decreasing synchronization interval improves precision but increases message load of the network [1].

In this study, we provide significant improvements to the TPSN. The first algorithm we propose divides the WSN into a number of clusters. TPSN can therefore provide synchronization among the clusterheads first then the clusterheads can synchronize asynchronously with their cluster members hierarchically providing different levels of synchronization. The second method we employ is based on chain-synchronization and the last method is based on adaptive selection of the interval to be used for synchronization based on the last synchronization results. We apply all of the foregoing in a sample WSN using the ns2 simulator and compare the results obtained by each method. Section II provides the necessary background, Section III details the three enhancement methods to the TPSN and finally the conclusions drawn are given in Section IV.

978-1-4244-2881-6/08/$25.00 ©2008 IEEE

## II. Background

TPSN is a Sender-Receiver based time synchronization protocol for WSN. It has two main steps to synchronize the network as the *pair-wise synchronization* and *the network wide synchronization*.

### A. Pair-wise Synchronization

Let's have 2 nodes $i$ and $j$ which will have been synchronized and node $i$ starts the synchronization. Here are the steps to synchronize $i$ to node $j$:

1) Node $i$ prepares a synchronization pulse packet and starts message send process
2) Just before transmission on lowest possible network layer, the packet is timestamped with $T_1$ and delivered to the environment.
3) When packet is delivered to node $j$, it is timestamped with $T_2$ immediately at lowest possible network layer again.
4) Node $j$ then prepares a synchronization acknowledgment packet and starts message reply process
5) Like other message transfers, at lowest possible network layer, the packet is timestamped with $T_3$ and delivered.
6) Node $i$ receives the packet and timestamps again with $T_4$ for last time.
7) Finally, node $i$ calculates the clock offset and fixes its clock.

Assuming O is the offset to be corrected and it needs to be calculated when reply package comes and furthermore assuming there is no timestamping uncertainties, offset O can be calculated simply as follows [3]:

$$O = \frac{(T_2 - T_1) - (T_4 - T_3)}{2} \quad (1)$$

After computing the offset, the clock difference is know between node i to node j. It depends on the implementation to change local clock using the offset or keep the offset in a separate place and use it when needed.

### B. Network-wide Synchronization

TPSN builds a spanning tree where each node knows the level of itself and the parent. Level 0 is assigned to one node which is named as root node. This node has the responsibility to build tree by triggering level setting phase. To start the tree construction, root node sends a level discovery packet with its level 0 in it. When all the one hop neighbors receive this packet, they set their level to 1, parent to 0 and send another level discovery packet with the level 1 in it. However, before sending another packet, each node waits for random time to avoid collisions. This process is done all other nodes. In summary, when a node receives a level discovery packet, it sets its level to the level one more in the packet and sets the sender node as its parent.

After a time, if a node could not get any level discovery packet, it sends a level request packet. When its neighbors receive this level request message, they reply this message with their level in it. Then the node sets it level to minimum received level + 1 and sets it parent to the sender of minimum level.

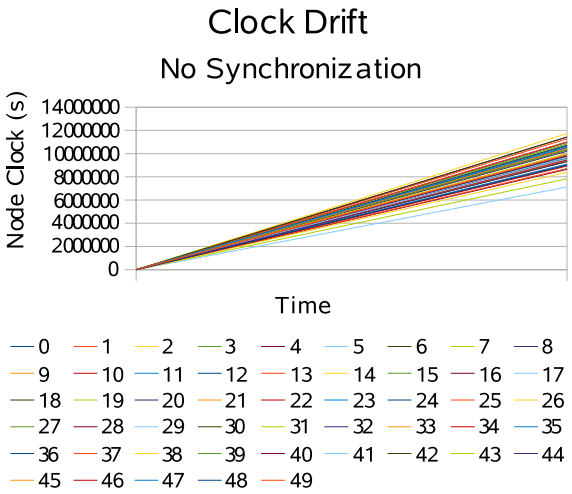


Fig. 1. Clock Drift On WSN

Synchronization is done for each node periodically by each node sending a sync pulse packet to its parent periodically. When a node cannot get a synchronization acknowledgment packet from its parent, that may mean that its parent it dead so it sends a level request packet to set a new level itself.

## III. Enhancements

First, in order to analyze enhancements done over TPSN, a simulation of a wireless network with 50 nodes is prepared using the $ns2$ simulator. All nodes are located in a 1,000x1,000 units area randomly using uniform distribution. To generate clock drifts for nodes, normal distribution is used with parameters $\mu$=1 and $\sigma^2$=0.1. In Fig 1, there is an obvious clock drift. This will be used as a reference result before synchronization is done.

Simulation duration will be 10,000,000 seconds and periodic synchronization interval will be 100,000 seconds. In Fig 2, the structure after level discovery phase can be seen. There are 14 levels in this network. Each row represents the level in TPSN protocol not the physical location of the node. Also, several colours are used to distinguish clusters, which will be explained later.

### A. Reference TPSN Results

In this phase, TPSN is directly applied to the network. Fig 3 shows the results of TPSN applied with no change using a sync interval of 100,000 seconds. The improvement in synchronization can be seen easily. However, there is still a considerable clock difference on the graph. Decreasing the interval will improve the synchronization but it will also increase the message load on the system. Therefore, we will look into ways to increase precision by not increasing message traffic too much.

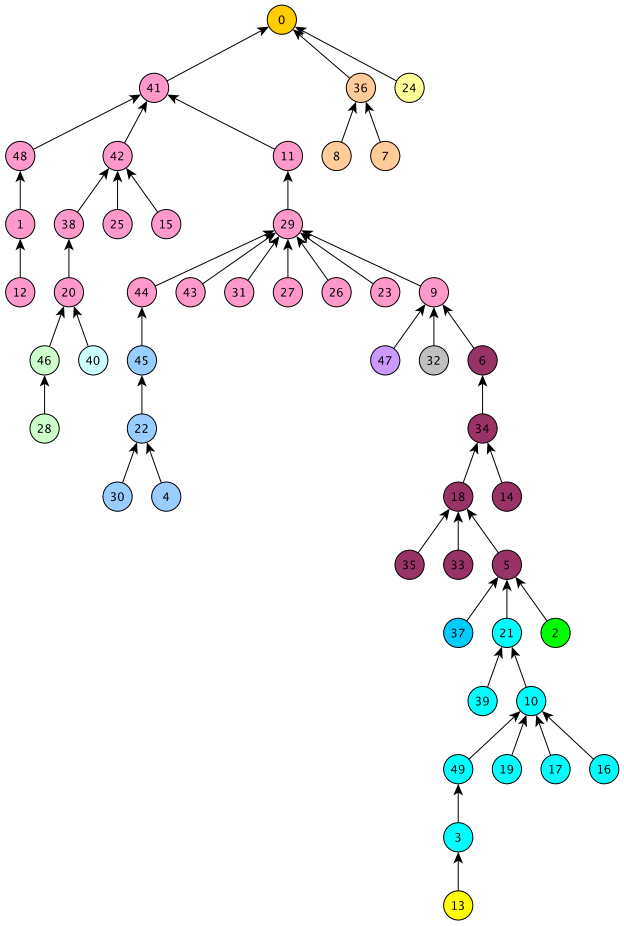


Fig. 2. WSN Hierarchy Used In Simulation

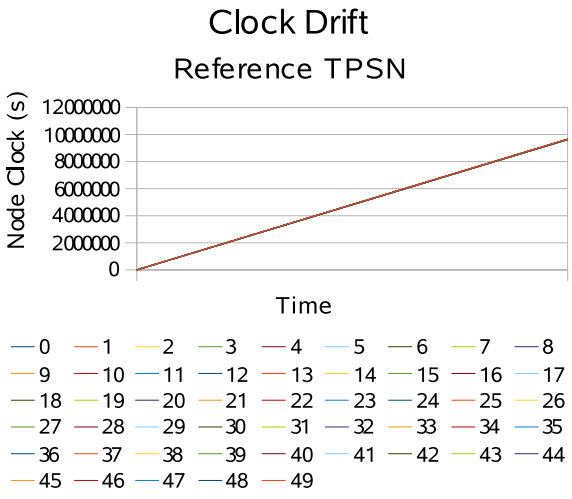


Fig. 3. Clock Drift On WSN - Reference TPSN

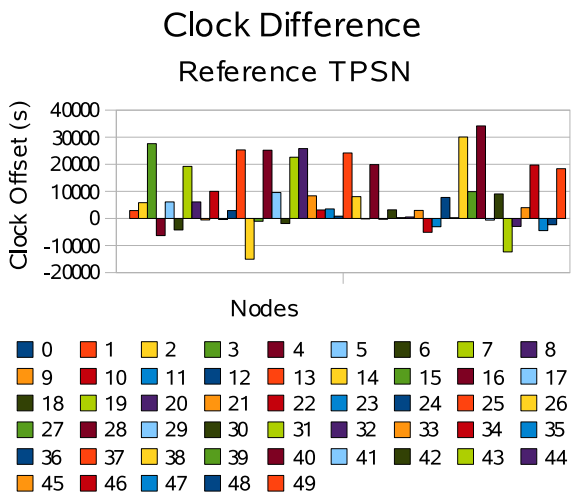


Fig. 4. WSN - Reference TPSN

### B. *Use of Clusters*

In this method, A spanning tree which is clustered using a level depth 4 will be used. This means node level 0-3 will be in a cluster, level 4-7 will be in another cluster and so on. This clustering algorithm simply assigns a group number to a node using its level and the depth number 4 is arbitrarily selected. Any other depth value or clustering algorithms could also be used. In Fig 2, each color represents a cluster. As an exception, root node will be left alone as another cluster at the top. Clustering itself is not enough for an improvement in synchronization. Because TPSN has only one hop message traffic, there needs to be a change in behavior between cluster synchronization. For clusters, root nodes of the clusters will be used in synchronization which means child cluster root sends a message for synchronization to parent root leaf. That leaf transfers the request to its parent and others also do this until the message reaches the parent cluster root node. That parent cluster root node then sends the time information back using the same path to the requested node in the child cluster. For the child and root node, there is no change in the TPSN behavior. For intermediate nodes, it is more reasonable to update intermediate node clocks also because there will be no message overhead. However, this needs temporary package modifications at intermediate levels.

*1) Chain Synchronization:* Instead of leaving intermediate node clocks unaffected, it is possible to update all the nodes on the path between cluster roots. Here is a simple scenario to understand how chain synchronization works. In this scenario, Node $i$ is the child, requesting synchronization. Node $j$ is the intermediate node and node $k$ is the root to be synchronized.

When node $i$ starts the synchronization, it asks for the time to its parent, which is node $j$. Node $j$ answers this request packet with acknowledge packet however, using the chain synchronization, forwards the synchronization request to node $k$. Before forwarding the request packet, node $j$ has three

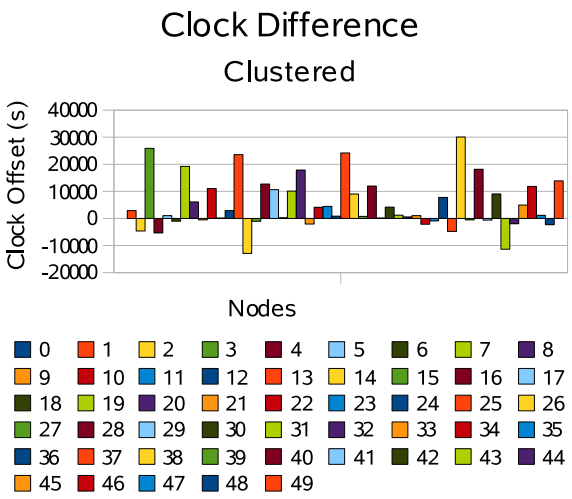


Fig. 5. WSN - Clustered TPSN

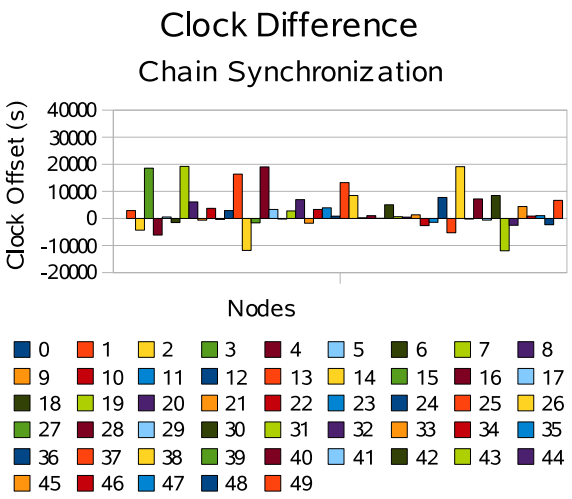


Fig. 6. WSN - Chain Synchronization For All

important jobs to do.

1) Store requester node's address: Because TPSN tree structure is directed and only the children know their parent. Therefore node $j$ stores node $i$ as requester node.
2) Store requester's sent time: This is crucial for TPSN offset calculation and it will be used when sending acknowledge packet to node $i$
3) Replace requester's sent time in the packet: To update intermediate node $j$'s clock, new TPSN packet must be prepared. Therefore, requester's sent time information is replaced with the node $j$'s sent time as required by TPSN.

When node $k$ receives the forwarded packet, it is just a TPSN packet from node $j$. So, node $k$ replies request with its time information. When node $j$ receives the packet, this packet can be used to update node $j$'s clock and so, node $j$ updates it clock. However, after updating the clock, node $j$ replaces requester's sent time information with the original one, which is stored before forwarding the request and forwards acknowledge packet to node $i$. With this packet, node $i$ updates its clock correctly.

The assumption here is that TPSN only uses 4 timestamps to synchronize the clocks so the time that passes on the parent node logically is not important. It can be seen that node $j$'s request forwarding and acknowledge waiting times can be considered as node $j$'s internal process. In this way, what is done here is actually similar to TPSN functionality.

Fig 5 shows that now the network is better synchronized with a low message overhead compared to reference TPSN synchronization in Fig 4.

### C. Chain Synchronization For All

The network is better synchronized when chain synchronization is performed for cluster root synchronization. This operation can be done for all nodes but if it is implemented in the same way as done for clusters, nodes with low level will be too busy for synchronization, especially the root. That might reduce precision because they cannot answer synchronization requests when doing another chain synchronization. It is possible to do chain synchronization for all request by implementation but that's not wanted because it will also increase message traffic a lot. Therefore, a simple solution may prevent this problem. If a node is in chain synchronization state and cannot answer child node's chain syncronization request, then it can reply request just like in TPSN. No chain request is required. TPSN like behaviour is kept here when an intermediate node is busy with chain synching.

The improvement in synchronization is depicted in Fig6. It can be seen that the message overhead here is minimum. Such results may be obtained with reference TPSN with reduced interval but but no interval change is done to test the methods to improve synchronization.

### D. Adaptive Interval

Most of the times, a clock difference up to some value is acceptable. Some nodes have a bigger drift and some nodes have smaller as seen in Fig 1. So, it is not an optimized solution to use same intervals for all the nodes. Therefore, we can gradually increase or decrease interval using the last synchronization results. 1% can be a reasonable step value. Results can be momentarily big or small so gradual changes is more reliable at this point. After some time, there will be a network which is optimized for message load. It is obvious that, if the expected value for clock difference is kept small, than it will increase the messages required. So, expected or the acceptable average clock difference should be carefully selected. In the example here, 10,000 seconds (1/10 of default sychronization interval) is selected as acceptable average clock difference. Fig 7 show that now the clock differences is closer and average clock difference is less. In this result, the total message count is reduced to the same value as reference TPSN used.

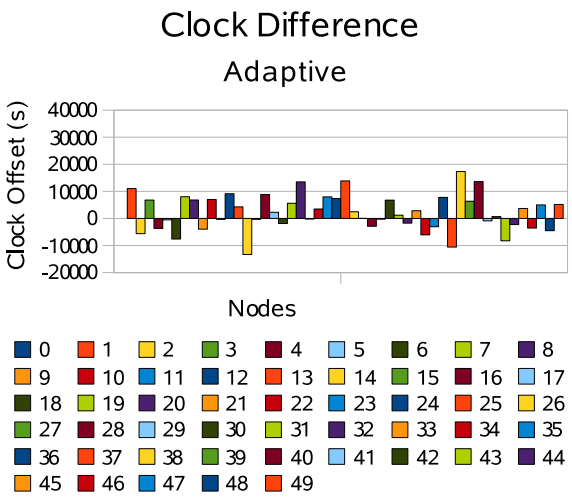


Fig. 7. WSN - Adaptive Synchronization

TABLE I
WSN - COMPARISON TABLE

| | Average | Standard Deviation | Message Count |
|---|---|---|---|
| Reference | 6721.78 (100%) | 11391.37 (100%) | 19512 (100%) |
| Clustered | 5022.81 (74.72%) | 9251.29 (81.21%) | 21096 (108.12%) |
| Chained | 2818.32 (41.93%) | 6990.67 (61.37%) | 21684 (111.13%) |
| Adaptive | 2139.02 (31.82%) | 6558 (57.57%) | 19852 (101.74%) |

## IV. CONCLUSION

Although TPSN produces a simple and effective way of time synchronization in the WSNs, the enhancement methods to TPSN proposed in this work provide reasonable precision improvements with low message overheads. Table I provides a comparison of all of the methods employed.

First improvement was making network hierarchy clustered and doing chain synchronization between cluster roots. As seen in the comparison table, both average clock difference and the standard deviation have decreased significantly. However, the message traffic has increased slightly which is expected.

Second enhancement was applying chain synchronization to all nodes. In this method, all nodes try to chain synchronize at first. However, it might not be possible to do it all the time. If it was not possible to synchronize in chain, parent node replies the request as similar to what reference TPSN does. Now, the synchronization was improved again significantly with a small message overhead using the clustered TPSN.

Last improvement was a tidying up method to remove extra message overhead. However, it was shown that it also enhanced the synchronization results. The reason behind this might be the balanced message traffic. The choice of adaptive interval offset parameter is very important and any improper value given can degrade network performance significantly. In conclusion, all the methods above provided significant improvements over TPSN. All can be implemented easily using TPSN, or any sender-receiver based synchronization protocol. That means TPSN is not mandatory here but results might vary for other protocols.

## REFERENCES


[1] B. Sundararaman, U. Buy, and A. D. Kshemkalyani, *Clock Synchronization for Wireless Sensor Networks: A Survey* March 2005

[2] F. Sivrikaya and B. Yener, *Time Synchronization in Sensor Networks: A Survey*, IEEE Network, July/August 2004.

[3] Ganeriwal, S., Kumar, R., and Srivastava, M. B. 2003. *Timing-sync protocol for sensor networks*. In Proceedings of the 1st international Conference on Embedded Networked Sensor Systems (Los Angeles, California, USA, November 05 - 07, 2003). SenSys '03. ACM, New York, NY, 138-149. DOI= http://doi.acm.org/10.1145/958491.958508

[4] Shujuan Chen, Adam Dunkels, Fredrik Osterlind, Thiemo Voigt, and Mikael Johansson. *Time synchronization for predictable and secure data collection in wireless sensor networks*. In Proceedings of The Sixth Annual Mediterranean Ad Hoc Networking Workshop (Med-Hoc-Net 2007), Corfu, Greece, 2007.

[5] Jeremy Elson, Lewis Girod, and Deborah Estrin. *Fine-grained network time synchronization using reference broadcasts*. In OSDI 02: Proceedings of the 5th symposium on Operating systems design and implementation, pages 147163, New York, NY, USA, 2002. ACM. doi: http://doi.acm.org/10.1145/1060289.1060304.

[6] Saurabh Ganeriwal, Ram Kumar, and Mani B. Srivastava. *Timing-sync protocol for sensor networks*. In SenSys 03: Proceedings of the 1st international conference on Embedded networked sensor systems, pages 138149, New York, NY, USA, 2003. ACM. ISBN 1-58113-707-9. doi: http://doi.acm.org/10.1145/958491.958508.

[7] Su Ping. *Delay measurement time synchronization for wireless sensor networks*. Technical report, June 2003.

[8] Ugo Buy Bharath Sundararaman and Ajay D. Kshemkalyani. *Clock synchronization for wireless sensor networks: A survey.* Technical report, Department of Computer Science, University of Illinois at Chicago 851 South Morgan Street Chicago, IL 60607, March 2005.

[9] Member-Qun Li and Member-Daniela Rus. *Global clock synchronization in sensor networks.* IEEE Trans. Comput., 55(2):214226, 2006. ISSN 0018-9340. doi: http://dx.doi.org/10.1109/TC.2006.25.

[10] Yanos Saravanos. *Energy-aware time synchronization in wireless sensor networks*. Technical report, Department of Computer Science, UNIVERSITY OF NORTH TEXAS, December 2006.

[11] Alan Mainwaring, Joseph Polastre, Robert Szewczyk, David Culler, John Anderson, *Wireless Sensor Networks for Habitat Monitoring, ACM International Workshop on Wireless Sensor Networks and Applications* (WSNA 02), Atlanta GA, September 28, 2002.

[12] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. *Wireless sensor networks: a survey*. Computer Networks (Elsevier), vol.38, pp.393-422, 2002.

[13] J. Elson and K. Romer, *Wireless Sensor Networks : A New Regime for Time Synchronization*. HotNets-I, Princeton, NJ, 2002.