International Journal of Future Generation Communication and Networking
Vol. 8, No. 4 (2015), pp. 57-68
http://dx.doi.org/10.14257/ijfgcn.2015.8.4.06

Adaptive Source Time Synchronization for Low-Duty-Cycle Wireless Sensor Networks

Tian Le

Institute of Electricity and Information Engineering, Beijing University of Civil Engineering and Architecture, Beijing, China, 100044
tianle@bucea.edu.cn

Abstract

Time synchronization is critical for most distributed systems, especially for low-duty-cycle Wireless Sensor Networks(WSNs). Low-duty-cycle WSNs make use of time synchronization in many contexts(scheduling and sleeping, TDMA, Event Identification, data fusion, etc). A novel adaptive source time synchronization algorithm designed for low-duty-cycle WSNs, namely ASTS, is presented in the paper. The algorithm can be used for a small Low-Duty-Cycle WSN to be synchronized to a common clock, or used for a giant Low-Duty-Cycle WSN to be synchronized distributed. To improve the synchronization accuracy, all nodes estimate their time drifts relative to their neighbors using Maximum Likelihood Estimation, and get be synchronized to a common clock or their heads based on a vector of time drifts carried by the reference packet sent by the reference node. Simulation shows that the algorithm drastically improves the synchronization accuracy and scalability, and is much more applicable for low-duty-cycle WSNs than other synchronization algorithms.

Keywords: *Time Synchronization, Gaussian distribution, Maximum Likelihood Estimation, Low-duty-cycle Wireless Sensor Networks.*

1. Introduction

Wireless sensor networks, which are composed of lots of low-cost, low-powered tiny wireless nodes[9,10], have been used for many long-term applications, such as environment monitoring[11], building protection and military surveillance[1,12] and supply chain and logistics management[7,8]. The node used in WSNs is so tiny that it always is powered by batteries and cannot be recharged. However, in order to reduce the operational cost, the node should be working as long time as it can. To bridge the gap between the limited energy and the lifetime, the node has to be working in a low-duty-cycle way, which means the node should be aware for a while to receive and send data and then falls to asleep for most times.

For a low-duty-cycle WSN, an agreed common clock is more critical: Firstly, with an agreed common clock, the nodes in the network can be scheduled to wake up at the same time, thus can get connection with their neighbors to setup a connected network. Secondly, with an agreed common clock, the nodes can use TDMA as MAC layer protocol, which will be more energy efficient. Thirdly, with an agreed common clock, the nodes which sense the data can identify the happening events, with the time as an important ID. In this way, the sensed data transmitted through the network can be fused and the number of transmitted data will be decreased significantly, which will reduce the energy consumption of the whole network.

The time synchronization error or clock drift in WSNs generally involves two parts: 1) clock offset, which is the clock time phase offset compared to one common absolute time and 2) clock skew, which is the clock frequency offset relative to a certain standard frequency. The second part is because the imperfections in the quartz crystal and the environmental conditions cause different clocks to run at slightly different frequencies. In fact, the effect of clock skew is the main reason why clock offsets keep drifting away.

Although lots of time synchronization algorithms proposed specially for WSNs have been designed, they are not suitable for low-duty-cycle WSNs because of the following reasons: First, most algorithms need the sensor nodes to exchange time information with their neighbors frequently, which will be energy-consumption for tiny sensor nodes. Second, most algorithms does not take the sleeping situation of the sensor nodes into account, and the result is lots of sensor nodes lose their synchronization due to the accumulative time error.

We propose a time synchronization algorithm applicable for low-duty-cycle WSNs in this paper. In our algorithm, the network is not needed to create a spanning tree or to be organized hierarchically. The nodes can get their clock drift estimations through neighborhood information exchanged with their neighbors using piggybacking or overhearing methods. Once a reference node, usually the Sink or the Workstation for a small wireless sensor network, or a set of selected reference nodes for a giant wireless sensor network, send(s) out a reference packet, the nodes can synchronize themselves to the reference node based on a common clock and a vector of time drifts carried by the reference packet hop by hop. If by any chance one node lost its synchronization, it can adjust its clock using MLE method to get be resynchronized.

The rest of the paper is organized as follows: related works are presented in section 2, section 3 describes the network model and the algorithm in detail, in section 4 we show the performance of the algorithm via simulation, and section 5 is about our conclusion and future works.

2. Related Works

Lots of time synchronization algorithms have been proposed for WSNs. As for RBS[6], a reference message is broadcasted by the reference node to its neighbors without containing the local time clock. The receivers after receiving the reference packet exchange their recorded time and synchronize with each other. In this way, the reference's non-determinism will be eliminated, but at the cost of additional messages. PBS[2] is another kind of synchronization algorithm using pair time information. PBS allows a sensor to synchronize itself by eavesdropping timing messages from a neighboring two-way message exchange. In a one-hop sensor network a single PBS message exchange between two neighboring nodes would synchronize all nodes, thus significantly reducing the communication overhead compared with RBS. In TPSN[14], a hierarchical structure will be created first before the procedure of time synchronization. Each node gets synchronized by exchanging two synchronization messages with its reference node. TPSN does not estimate the clock drift of nodes, which make its accuracy lower compared with other algorithms.

FTSP[15] is the de facto standard time synchronization protocol in sensor networks. In FTSP, the reference packet is broadcasted thoroughly into the network periodically by a reference node elected dynamically, which contains the value of its current time information. Each node who receives the packet can get the linear relationship between its own hardware clock with the received clock using least-squares regression. Then the receivers can predict future clock values and achieve the network-wide synchronization. However FTSP is based on reference packets periodically broadcasted, which may not be available for low-duty-cycle wireless sensor networks.

Gradient Time Synchronization Protocol (GTSP) [16] is another kind of time synchronization protocol which optimizes local skew in wireless sensor networks. GTSP is totally decentralized and all nodes are synchronized to their neighbors, which makes

GTSP not applicable for the low-duty-cycle WSNs where their neighbors may be unavailable.

Kasim Sinan Yildirim introduces EGSync and FCSA in [3] and [4] respectively. In EGSync, each sensor node synchronizes itself to a reference node by using time information flooded by this node, as well as synchronizes itself to its neighboring nodes by employing an agreement algorithm. In FCSA, every node is forced to run at the same speed by employing an agreement algorithm between neighbors and is synchronized to a reference node that floods stable time for the whole network using slow-flooding method. But similar to GTSP, both algorithms are not applicable for the low-duty-cycle WSNs.

3. Model and Analysis

The wireless sensor network is modeled as a graph G={V, E}, where V={1, …, n} represents the set of nodes of the network and E=VxV represents the bi-directional communication link between the nodes. All nodes that can communicate with a certain node $u \in v$ directly are u's neighbors, which can be denoted as $\mathrm{N}(u) = \{v \in V \mid (u,v) \in E\}$, and $|\mathrm{N}(u)|$ is used to denote the number of node u's neighbors.

The basic steps of our algorithm are as follows: All nodes in the network estimate their time drift relative to their neighbors and the time drift relative to the reference node. For node u, the set of time drifts relative to its neighbors will be stored as $\Theta(u) = \{\theta_{uv} \mid v \in \mathrm{N}(u)\}$, and the time drift relative to the reference node will stored as θ_u . The reference node, which may be the Sink node or the workstation for a small Low-Duty-Cycle WSN, or a set of reference nodes chosen randomly for a giant Low-Duty-Cycle WSN, broadcast(s) the reference time packet(s) periodically, the reference packet contains a common clock $T_{reference}$ and a vector of time drifts V_{drift}, which will be empty at first. Every node who receives the reference packet from its neighbor, will adjust its time clock according to the common clock $T_{reference}$ and the vector V_{drift}, then append its time drift relative to the neighbor who delivers the reference packet to it at the end of the vector V_{drift}, and then floods the reference packet out. For example, node j receives the original time reference packet, which contains a common clock and a vector of time drifts $\{T_{reference}, V_{drift}(\theta_{ab}, \theta_{be}, ..., \theta_{hi})\}$ from its neighbor node i, node j will calculate its time based on $T_{reference}$ and V_{drift}, append the time drift θ_{ij} at the end of V_{drift}, which will be changed as $V_{drift}(\theta_{ab}, \theta_{be}, ..., \theta_{hi}, \theta_{ij})$, then broadcasts the time reference packet out.

3.1 How to Get $\Theta(u)$

As described in TPSN, for two neighboring nodes, the initiator can synchronized itself to its neighbor in this way(as shown in Figure 1): The initiator sends a initial pulse containing time T1 to the neighbor, the neighbor receives this pulse at time T2, the deference between T1 and T2 contains the clock drift of A to B t_{ab} and the propagation delay d_{ab}, after a random time interval, the neighbor sends back an acknowledge packet containing T1, T2 and T3, and once upon the initiator receives the packet at time T4, it can calculate the time drift and the propagation delay as follows:

$$t_{ab} = \frac{(T2 - T1) - (T4 - T3)}{2} \quad (1)$$

$$d_{ab} = \frac{(T2 - T1) + (T4 - T3)}{2} \quad (2)$$

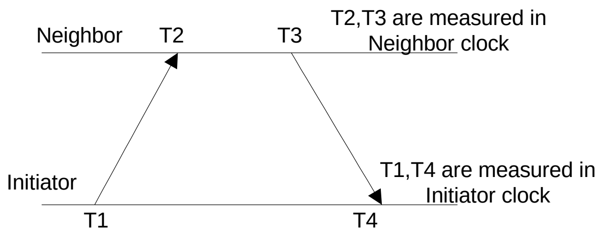
Neighbor
T2
T3
T2,T3 are measured in Neighbor clock
Initiator
T1
T4
T1,T4 are measured in Initiator clock

Figure 1. Two-way Time Information Exchange

The propagation delay d_{ab} is too small and can be neglected in time synchronization, so we want to know the value of t_{ab}. The time drift t_{ab} calculated in this way will not be accurate enough. If we want to get an accurate t_{ab}, we can let node A and B measure t_{ab} many times, denoted as: $(t_{ab}^{(1)}, t_{ab}^{(2)}, ..., t_{ab}^{(n-1)}, t_{ab}^{(n)})$.

Noh[5] has proved that the time phase offset in this two-way time information exchange is a Gaussian Delay Model, then according to maximum likelihood estimation(MLE), the likelihood function is:

$$L(t_{ab}^{(1)}, t_{ab}^{(2)}, ..., t_{ab}^{(n-1)}, t_{ab}^{(n)} \mid \theta_{ab}, \sigma_{ab}^2) = (\frac{1}{2\pi\sigma_{ab}^2})^{n/2} \exp(-\frac{\sum_{i=1}^{n}(t_{ab}^{(i)} - \overline{t_{ab}})^2 + n(\overline{t_{ab}} - \theta_{ab})^2}{2\sigma_{ab}^2})$$
(3)

Differentiating the log-likelihood function, let $\frac{\partial L(t_{ab}^{(1)}, t_{ab}^{(2)}, ..., t_{ab}^{(n-1)}, t_{ab}^{(n)} \mid \theta_{ab}, \sigma_{ab}^2)}{\partial \theta_{ab}} = 0$, we can get the estimation of the expected value of time drift θ_{ab} as:

$$\theta_{ab} = \overline{t_{ab}} = \frac{\sum_{i=1}^{N} t_{ab}^{(i)}}{N}$$
(4)

For a tiny and energy-constrained sensor node, exchanging two-way time information frequently will be an energy exhausting procedure. Fortunately, we can let the node piggyback its time information while sending out data or broadcasting packets, and overhear its neighbor's answer to get time information. All the node has to do is to add timestamps to the packet it sends or receives, and eavesdrops its neighbor's answer and record the arriving time, calculates the time drift of this moment, recalculates the expected value θ_{ab}. Assuming the node has got a drift $t_{ab}^{'}$ at this moment, and the stored estimated expected value is θ_{ab}^{old} and the times of calculation is N, then the new estimation of the expected value of the time drift will be:

$$\theta_{ab}^{new} = \frac{N\theta_{ab}^{old} + t_{ab}^{'}}{N+1}$$

3.2 How to Synchronize to a Common Clock in a Small Low-duty-cycle WSN

For some kinds of applications, building surveillance or smart living as examples, a small Low-Duty-Cycle WSN containing dozens or hundreds of nodes will be applicable. In order to synchronize all nodes of this kind of WSNs to a common clock, the reference node, usually to be the Sink or the Workstation, broadcasts a reference packet containing a common clock and a vector of time drifts periodically. The packet will be flooded in the network hop by hop. When a node, jth node as an example, receives the reference packet from its neighbor node i, node j will get a common clock $T_{reference}$ and a vector of time difts $V_{drift}(\theta_{ab}, \theta_{be}, ..., \theta_{hi})$. $T_{reference}$ is the common clock broadcasted by the reference node, vector $V_{drift}(\theta_{ab}, \theta_{be}, ..., \theta_{hi})$ is a series of time drifts appended by the nodes on the path from the reference node to node j.

Because each node on the path works independently, the time drift t_{xy}(x, y belong to the path) is independent with each other. We assume the total time drift from the reference node to node j is $t_j = t_{ab} + t_{bc} + ... + t_{hi} + t_{ij}$. Since t_{xy}(x, y belong to the path) is a Gaussian delay model, the distribution of t_j is a normal distribution, whose estimation of the expected value will be:

$$\theta_j = \theta_{ab} + \theta_{bc} + ... + \theta_{hi} + \theta_{ij} \quad (5)$$

Thus node j can use $T_{reference}$ and θ_j to synchronize to a common clock by setting its time as $T_{reference} + \theta_j$, node j records θ_j as its total time drift relative to the reference node simultaneously.

However, in a low-duty-cycle WSN, a node may lose its synchronization due to the drop of the reference packet, transmission contention, excessive sleeping cycle, etc. For an entire synchronization cycle a node did not receive any reference packets, the node can judge that it has lost synchronization. In this kind of situation, the lost node, denoted as j, will send a synchronization request to its neighbors, try to synchronize to one of its neighbors. Once upon receiving an answer packet containing a reference time $T^i_{reference}$ from node i, one of its neighbors, node j will synchronize to node i by setting its time as $T^i_{reference} + \theta_{ij}$.

If a node wakes up while all its neighbors are falling into sleeping, the node will neither can be synchronized to the reference node, nor can be synchronized to its neighbors. In this situation, the node will synchronize itself to the reference node by setting its time as local time minus θ_j, namely $T_{localtime} - \theta_j$. Using this way, the node can be self-synchronized.

That is still possible that self-synchronization fails. In this situation the node will keep awake for several synchronization cycles, try to synchronize to the reference node. Only if the node is synchronized, can it fall into asleep again.

The pseudo-code of the synchronization algorithm is shown as follows (node j as an example):

```
1: Initialization $\Theta(j)$ by piggybacking and overhearing;
2: After receiving a reference packet:
3:     calculate $\theta_j$;
4:     set local time as $T_{reference} + \theta_j$;
5: If not receive any reference packets for a synchronization cycle:
6:     Send a synchronization request to neighbors;
7:     If receive a synchronization replay from one of neighbors, node i:
8:             set local time as $T^i_{reference} + \theta_{ij}$;
9:     Else
10:            self-synchronized by set time as $T_{localtime} - \theta_j$;
11:            If not synchronized yet:
12:                    Keep awake until be synchronized to the reference node;
13:            End If
14:    End If
15: End If
```

3.3 How to Synchronize to a Common Clock in a Giant Low-duty-cycle WSN

However, for some applications needing lots of nodes working in a cooperative way, such as environment monitoring or battlefield reconnaissance, the coverage area will be several square kilometers, the number of nodes will be thousands. In these kinds of WSNs, all nodes synchronized to a unique reference node simultaneously will be not applicable as the time consumption from the reference node to a remote node is so great that the accumulative deviation of the time drift is very huge, this remote node has lost its connection with the reference node before be synchronized.

During a period of a time synchronization process, in order to decrease the accumulative deviation of time drift, the length of the path from the reference node to the synchronized node should be limited. So the process of the synchronization will be proceeding in several areas or clusters distributed and simultaneously. Once upon the synchronization process begins, each sensor node will select itself as one of the reference nodes voluntarily with probability p, and then advertise it's time information and an empty vector of time drifts as the reference packet to all its neighbors. The advertisement will be limited to k hops away from the voluntary reference node to limit the size of this synchronization area. A sensor node who receives a reference packet will join the synchronization area covered by the reference node who originated this reference packet, synchronizes itself to the common clock carried by the reference packet using the way we described in Section 3.2, and discards all other reference packets received from other reference nodes subsequently. However, there has chance that a node does not receive any reference packets from volunteer reference nodes. So after waiting for time duration t(where t is the estimated time for the reference packet to be transmitted from the reference node to a node), if a certain node does not receive any reference packets from other reference nodes, it will select itself as a reference node forcibly and broadcast its time as the reference packet to all of its neighbors.

If a certain node acts as a reference node(voluntarily or forcibly) all the time, it will consume much more energy compare with ordinary nodes, which is harmful for the lifetime of the whole network. And if a reference node does not be changed all the time, the synchronized time of this synchronization region will vary from other nodes in other regions due to the time drifts of their reference nodes. So in order to balance the energy consumption of all nodes and the time drifts of different synchronization regions, the reference nodes should vary from different time synchronization processes.

The key parameters of the algorithm are the probability p to be a reference node and the hops k to limit the size of the synchronization region. Assuming the distribution of the sensor nodes in the covered region is a homogeneous spatial Poisson process with intensity λ, according to [17], the value of probability p will be:

$$p = \left[\frac{1}{3c} + \frac{\sqrt[3]{2}}{3c(2+27c^2+3\sqrt{3}c\sqrt{27c^2+4})^{1/3}} + \frac{(2+27c^2+3\sqrt{3}c\sqrt{27c^2+4})^{1/3}}{3c\sqrt[3]{2}} \right]^2 \quad (6)$$

Where $c = 3.06\alpha\sqrt{\lambda}$.

And the value of k will be:

$$k = \left\lceil \frac{1}{r}\sqrt{\frac{-0.917\ln(\alpha/7)}{p\lambda}} \right\rceil \quad (7)$$

where r is the communication radius of a sensor node, and α is a parameter we can predetermine before the network is deployed to control the size of the synchronization area. The smaller the value of α is, the bigger the synchronization area will be.

4. Simulation and Performance

We use NS2 to simulate our algorithms. According to [6], the frequency of the clock of a typical crystal-quartz oscillator commonly used in sensor networks varies up to 40 PPM, which means clocks of different nodes can drift as much as 40 μs in a second (or 0.144 s in an hour). So we add a time agent upon the node agent, the time agent will generate a time offset which is a normal distribution with expected value as 20 PPM.

At first, we simulate the algorithm with 500 nodes deployed in a 1kmX1km region randomly, which can be used as a typical environment surveillance scenario. The duty cycle of every node is ten seconds, the node will fall into asleep for about 10 minutes after working for a duty cycle. A node picked out randomly acts as the Sink, who broadcasts a time synchronization beacon every five working cycles. Each round of simulation lasts for 3 hours. We run our algorithm 50 times and get the average drift as the result. For a comparison, we run FTSP, FCSA as well. Table 1 shows the average time drift of three kinds of time synchronization, from which we can draw a conclusion that the performance of ASTS is similar to FCSA, but much better than FTSP for a short term working period.

Table 1. Average Time Drifts of Three Algorithms

	ASTS	FTSP	FCSA
Max. Time Drift	23.3μs	45.4μs	24.2μs
Average Time Drift	12.1μs	24.8μs	12.3μs

To observe the performance of three algorithms in a long term working situation, we prolong the simulation time to 6 hours, 12 hours, 24 hours respectively. Figure 2 plots the average time drifts of three algorithms while the simulation time is prolonged. The figure tells us that along with the increase of the working period, the average time drift of ASTS stays stable compared with FTSP and FCSA. From analysis we find that the increase of time drift in FTPS and FCSA is because some nodes lose their synchronization while the working period is prolonged.

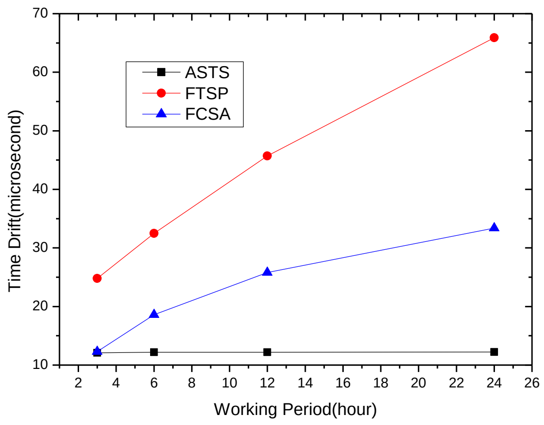

Figure 2. Average Time Drift

We also count the number of nodes whose synchronization is lost in three algorithms. Fig. 3 is the results of our algorithm, FTSP and FCSA. From which we can see that the average number of nodes which lose its synchronization in ASTS is quite small and stays stable compared with FTSP and FCSA increasing sharply for a long working period and low-duty-cycle working situation, which means our algorithm is more applicable for this kind of WSNs.

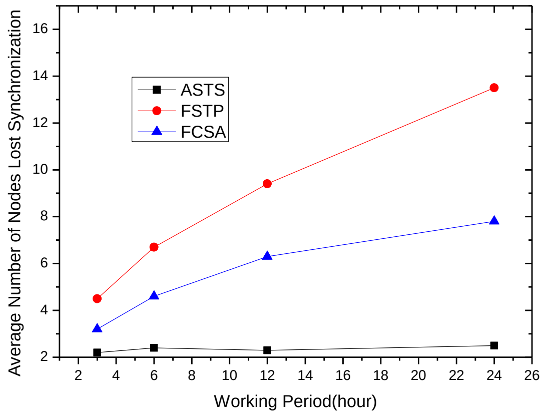

Figure 3. Average Number of Nodes Lost Synchronization

According to the common sense, the smaller the ratio between the time of staying awake and the time of falling into asleep within a working cycle is, the worse accuracy a time synchronization algorithm can get. For a typical environment surveillance or a building monitoring scenario, the waking time of a node may be only 10 minutes and then the node falls into asleep for a whole day. So we simulate three algorithms with the ratio of the waking time to the sleeping time varying from 1/60(10 seconds for working, 10 minutes for sleeping) to 1/720(1 minute for working, 12 hours for sleeping). Fig. 4 shows the increase of the average time drifts of three algorithms. For FTSP, the time drift increases exponentially while the ratio decreases. For ASTS and FCSA, the time drifts increase almost linearly, but ASTS's performance is better than FCSA's. Fig. 5 shows the average number of nodes who has lost their synchronization while the ratio is changed. From it we can learn that the performance of ASTS is the best among three algorithms.

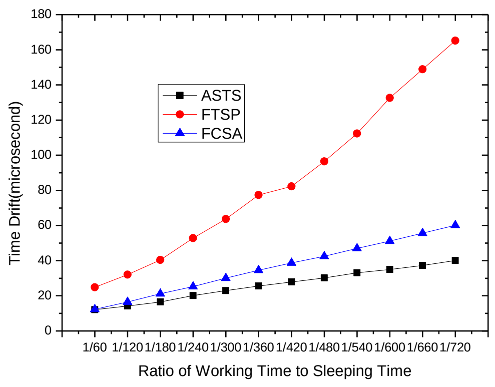

Figure 4. Average Time Drift while the Ratio of Working Time to Sleeping Time Changing

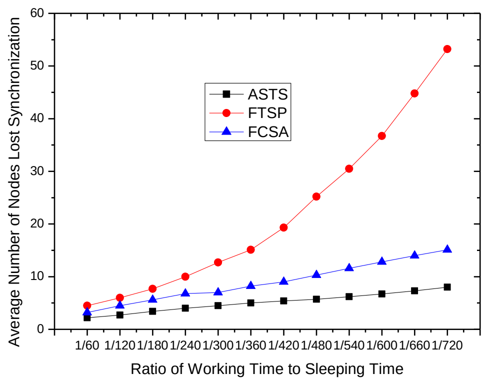

Figure 5. Average number of Nodes Lost Synchronization while the Ratio of Working Time to Sleeping Time Changing

We also simulate ASTS along with FTSP and FCSA in a giant wireless sensor network using JiST, which is a simulator similar to NS2, but can simulate a much bigger network. We assume that 100,000 nodes are deployed in a 10KM X 10KM region randomly, the communication radius of a node is 50m, the duty cycle is set to be 24 hours, the working period is 1 hour during each duty cycle. While in the working period, a node stays awake to collect data, communicate with its neighbors to get synchronized or transmit data. In other times, the node will fall into asleep to save energy consumption.

Table 2 shows the average time drifts of three algorithms. From which we can see that ASTS outperforms FTSP and FCSA heavily in maximum time drift and average time drift. The main reason is that in such a huge network and such a low duty cycle, the accumulative errors of FTSP and FCSA are much bigger than ASTS, while FTSP and FCSA are both synchronized to one reference node.

Table 2. Average Time Drifts of Three Algorithms in a Giant WSN

	ASTS	FTSP	FCSA
Max. Time Drift	0.34ms	2.51ms	1.93ms
Average Time Drift	0.21ms	1.89ms	1.57ms

Table 3 shows the average numbers of nodes which have lost their synchronization counted for three algorithms in a giant wireless sensor network. As for FTSP and FCSA, due to huge accumulative time error, up to 7.5% and 5.5% of all nodes have lost their synchronization, which maybe jeopardize the whole project. However while using ASTS, only 0.134% of nodes have lost their synchronization, the whole network will stay connected and can work cooperatively, the data can be sensed and collected in a right way.

Table 3. Average Number of Nodes Lost Synchronization in a Giant WSN

	ASTS	FTSP	FCSA
Number of Nodes	134	7458	5456

5. Conclusion

Time synchronization is critical problem for a low-duty-cycle WSN, but there has no dedicated time synchronization algorithm for this kind of WSNs. FTSP suffers from poorly synchronized accuracy and FCSA is unstable while the working period of the node is prolonged. In this paper we propose an adaptive time synchronization method, named as ASTS and designed specially for low-duty-cycle WSNs. In ASTS, all nodes will be synchronized to a reference node based on a common clock and a vector of estimated time drifts of nodes on the broadcasting path. Every node estimates its time drift relative to its neighbors based on maximum likelihood estimation via piggybacking and overhearing. So the longer the working time is, the more accurate of the time drift estimation is. Once upon losing its synchronization, a node can synchronize itself to its neighbors or get self-synchronized based on estimated total time offset. Simulation proves that ASTS is more applicable than FTSP and FCSA for low-duty-cycle WSNs. Now we are building our low-duty-cycle testbed network which containing 20 nodes. Our next work is to implement our algorithm in the testbed network to verify it.

Acknowledgements

We would like to thank for the support by Beijing Municipal Commission of Education under grant number KM201310016002.

References

[1] T. He, S. Krishnamurthy, L. Luo, T. Yan, L. Gu, R. Stoleru, G. Zhou, Q. Cao, P. Vicaire, J. A. Stankovic, T. F. Abdelzaher, J. Hui and B. Krogh, “VigilNet: An Integrated Sensor Network System for Energy-Efficient Surveillance”, ACM Transactions on Sensor Networks, vol.2, no.1, **(2006)**.
[2] K.-L. Noh, E. Serpedin and K. A. Qaraqe, “A new approach for time synchronization in wireless sensor networks: Pairwise broadcast synchronization”, IEEE Transaction on Wireless Communications, vol.7, no.9, **(2008)**.
[3] K. S. Yildirim and A. Kantarci, “External Gradient Time Synchronization in Wireless Sensor Networks”, IEEE Transactions on Parallel and Distributed Systems, vol.25, no.3, **(2014)**.
[4] K. S. Yildirim and A. Kantarci, “Time Synchronization Based on Slow-Flooding in Wireless Sensor Networks”, IEEE Transactions on Parallel and Distributed Systems, vol.25, no.1, **(2014)**.
[5] K.-L. Noh, Q. M. Chaudhari, E. Serpedin and B. W. Suter, “Novel Clock Phase Offset and Skew Estimation Using Two-Way Timing Message Exchanges for Wireless Sensor Networks”, IEEE Transactions on Communications, vol.55, no.4, **(2007)**.
[6] Y.-C. Wu, Q. Chaudhari and E. Serpedin, “Clock Synchronization of Wireless Sensor Networks: Message exchange mechanisms and statistical signal processing techniques”, IEEE Signal Processing Magazine, **(2011)**.
[7] J. Wang, X. Zhang, X. Hu and J. Zhao, “Cloud Logistics Service Mode and its Several Key Issues”, Journal of System and Management Sciences, vol.4, no.1, **(2014)**.
[8] M. A. Rad, F. Khoshalhan and M. Setak, “Supply chain single vendor – Single buyer inventory model with price-dependent demand”, Journal of Industrial Engineering and Management, vol.7, no.4, **(2014)**.
[9] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler and K. Pister, “System architecture directions for networked sensors”, Proc. of Architectural Support for Programming Languages and Operating Systems-IX, **(2000)**; Cambridge, USA.
[10] E. Shih, S. Cho, N. Ickes, R. Min, A. Sinha, A. Wang and A. Chandrakasan, “Physical layer driven protocol and algorithm design for energy-efficient wireless sensor networks”, Proceeding of MobiCom, **(2001)**; Rome, Italy.
[11] L. Selavo, A. Wood, Q. Cao, T. Sookoor, H. Liu, A. Srinivasan, Y. Wu, W. Kang, J. Stankovic, D. Young and J. Porter, “LUSTER: Wireless Sensor Network for Environmental Research”, Proceeding of SenSys, **(2007)**; Sydney, Australia.
[12] N. Xu, S. Rangwala, K. K. Chintalapudi, D. Ganesan, A. Broad, R. Govindan and D. Estrin, “A Wireless Sensor Network for Structural Monitoring”, Proceeding of SenSys, **(2004)**; Baltimore, USA.
[13]J. Elson, L. Girod and D. Estrin, “Fine-grained network time synchronization using reference broadcasts”, Proceeding 5th Symp. Operating System Design and Implementation, **(2002)**; Boston, USA.
[14] S. Ganeriwal, R. Kumar and M. B. Srivastava, “Timing-sync protocol for sensor networks”, Proceeding of SenSys, **(2003)**; Los Angeles, USA.

[15] M. Maroti, B. Kusy, G. Simon and A. Ledeczi, “The Flooding Time Synchronization Protocol”, Proceeding of Sensys, **(2004)**; Baltimore, USA.
[16] P. Sommer and R. Wattenhofer, “Gradient Clock Synchronization in Wireless Sensor Networks”, Proceeding of IPSN, **(2009)**; San Francisco, USA.
[17] S. Bandyopadhyay and E. J. Coyle, “An Energy Efficient Hierarchical Clustering Algorithm for Wireless Sensor Networks”, Proceeding of INFOCOM, **(2003)**; San Francisco, USA.

Authors

Tian Le, he received the B. S. degree in electronic precision machinery from Xidian University in 1994 and the Ph.D. degree in computer science and technology from the National Key Lab of Network and Switching of Beijing University of Posts and Telecommunications in 2007. He is currently an Assistant Professor of Computer Science in the College of Electrical and Information Engineering of Beijing University of Civil Architecture and Engineering, Beijing. His research interests are mainly in wireless communications, Internet of Things, wireless sensor networks.