International Journal of Multimedia and Ubiquitous Engineering
Vol.9, No.6 (2014), pp.283-300
http://dx.doi.org/10.14257/ijmue.2014.9.6.28

Diffusion-based Time Synchronization in Large-Scale Distributed Sensor Networks

Bongkyo Moon

Department of Computer Science and Engineering, Dongguk Univ-Seoul, Korea
bkmoon@dongguk.edu

Abstract

In this paper, the works on time synchronization in wireless sensor networks are investigated step by step. In particular, the diffusion-based algorithms for global synchronization in large-scale distributed sensor network are intensively focused. We thus propose a fast-converged asynchronous diffusion synchronization scheme in order to improve the performance of the asynchronous averaging diffusion method, and then prove its convergence mathematically. The evaluation results and discussions for the proposed scheme are also presented with simulation study. We eventually show that the proposed scheme converges a little faster than the asynchronous averaging diffusion method.

Keywords: *large-scale, distributed, sensor network, time synchronization, diffusion*

1. Introduction

Recently, small smart devices start to be embedded into the various environments in order to monitor the events occurred in the areas such as homes, plantations, oceans, rivers, streets, and highways. These tiny and low power devices which enable sensing and communication tasks have made sensor networks emerged. Typically, wireless sensor networks (WSNs) are a special type of ad-hoc networks, where wireless sensor nodes get together and spontaneously form a network without any infrastructure. Due to the absence of infrastructure such as router in traditional network, nodes in a sensor network have to cooperate for communication by forwarding each other's packets from a source to its destination. Thus, this yields a multi-hop communication environment. Since a sensor system with local clock is not capable of coordinated operation and data synthesis for future predictions, moreover, it is required to globally synchronize the clocks of all the nodes in the whole network. That is, all the clocks need to have approximately the same reading at a global time point, irrespective of their relative distance.

In general, a distributed system consists of a collection of distinct processes running concurrently in multiple nodes where each node uses a local clock for handling the time without global clock. Hence, each process in different nodes should use their local clock, and the frequency of these clocks may differ from each other and thus this finally causes drift in time over a period of time [1, 9, 11]. Therefore, these clocks may not remain synchronized all the time. Typically, time synchronization is a critical middleware service for many applications and operating systems in large-scale distributed systems [6, 9]. There are actually certain applications which need the time synchronization in distributed systems such as banking applications, database queries and real time applications. Hence, time synchronization in distributed system is so important because time-based events can be handled only if all the nodes in distributed system share a common notion of time. Since the clocks in distributed system do not remain well synchronized without periodic synchronization, the clocks on the nodes must resynchronized periodically by using the clock synchronization algorithm in order to maintain the global time.

Depending on the certain applications in distributed system, we need to know the time and the date at which an event happened on a particular node, and the relative ordering among events and the time interval between two events that occurs on different nodes [10]. Specifically, the knowledge of time between the sensor nodes in WSNs is essential that detect the events such as target tracking, speed estimating, and ocean current monitoring. Hence, the sensed data often loses valuable context without accurate time information. With time synchronization, voice and video data from the different sensor nodes can be fused and displayed in a meaningful way at the sink. Particularly, the common services in WSNs, such as coordination, communication, security, power management and distributed logging depend on the global time scale [41].

Consequently, global time synchronization is very important in large-scale distributed system environments like WSNs. Time synchronization in a WSN aims at providing a common time-scale (frequency and phase) for local clocks of nodes in the network. Conventional distributed algorithms for time synchronization mainly depends on the achievement and maintenance of a common time-scale for all the nodes in the network. These algorithms exchange the local time information through packets carrying a time-stamp. Specifically, distributed time synchronization is to provide correction factors to each node in the network, and thus enable the node to convert its own clock value to that of a unique common global clock [30]. However, traditional distributed algorithms cannot be considered for problems due to their unique characteristics, especially the severe resource constraints. Furthermore, most of existing synchronization methods are not scalable for very large networks since they use global time information sent to all the nodes. The initializing node may encounter failure and, thus, the approach is not fault-tolerant. All the nodes to be synchronized together must execute the clock update approximately at the same time, which may be too difficult in a large-scale distributed system [29].

Until now, meanwhile, diffusion-based algorithms have been some of the most popular methods for dynamic load balancing in traditional distributed systems [12, 14-16]. In diffusion-based load balancing methods, each computer can give its load to other computers or take load from them if it is under-loaded. Cybenko [7] and Boillat [8] analyzed the diffusion method in load balancing. They identified the sufficient and necessary conditions for convergence. The time complexity of the diffusion method was also analyzed in [29]. It is known, however, that this type of algorithm can suffer from slow convergence.

In this paper, therefore, we investigate the asynchronous diffusion methods and then propose a fast-converged asynchronous diffusion scheme for improving the performance of this kind of methods. This paper is organized as follows: In Section 2, the backgrounds and related works on time synchronization are introduced. In Section 3, we also investigate the diffusion-based algorithms and introduce rate-based asynchronous diffusion method. Fast-converged asynchronous synchronization scheme is proposed in Section 4. We also give evaluation results and discussions in Section 5. Finally, the conclusion is given in Section 6.

2. Backgrounds and Related Works

2.1. Clocks and Synchronization

For the general system model of clock synchronization algorithms, in this subsection, we consider a set of distributed sensor nodes interconnected by a wireless sensor network that can have different characteristics. The clock is essentially a timer that counts the oscillations of a quartz crystal with a particular frequency. Every sensor node should maintain its own clock which is the only notion of time. Each sensor node is mostly equipped with a hardware oscillator in order to assist its clock, which is a collaboration of hardware and software parts. Then, we can represent the clock for node *A* at real-time *t* by an approximation $C_A(t)$. The clock difference between two sensor nodes is normally referred as the offset error. There are actually three reasons for the node clocks to represent different times: First, the node clocks might be started at different times, second, due to the slightly different frequencies among the quartz crystals operating on the sensor nodes, the clock values might be gradually diverge from each other (termed as the skew error), or third, due to the aging or ambient conditions such as temperature, the clock frequency of the nodes can change differently over time (termed as the drift error) [34].

The synchronization on a wireless sensor network is basically to equalize the clocks of the *n* sensor nodes on it. The synchronization can be either global or local according to the range it covers. That is, it depends on trying to equalize $C_i(t)$ for all $i = 1::n$ or for some set of the nodes within the same spatial locality. However, equalizing just the offsets is not enough for effective clock synchronization since the clocks will drift away afterwards. Therefore, the clock rates as well as offsets should be equalized together, otherwise the offsets should be repeatedly corrected in order to keep the clocks synchronized over a time period [25].

This kind of strict definition of synchronization actually can be relaxed to rather loose degrees according to the characteristic of an application. In general, the synchronization mechanism can be classified into three basic types [21]. First mechanism is just to compare the local clocks for order of events or messages in order to tell whether a particular event has occurred before or after another event. This algorithm is just ordering rather than synchronization and an example to this type of synchronization is given in [23]. Second mechanism is for all nodes to maintain a clock that is synchronized to a reference clock in the network. Thus, it is able to preserve a global timescale throughout the network. The synchronization scheme in [21] conforms to this model. Third mechanism is for all nodes to maintain information about the relative drift and offset of their clock to other clocks in the network in addition to running their local clocks independently. Thus, at any instant, the local time of the node can be calculated from this kind of information. Most of the synchronization schemes for sensor networks use this model [20, 24].

2.2. Related Works

Until now, a lot of mechanisms to synchronize the local clocks of the nodes in WSN have been proposed [2-4, 13, 27, 31-33, 37-38, 42-45, 47-48]. In this subsection, we extensively investigate the major approaches among those mechanisms for time clock synchronization in WSNs. The algorithm of Lamport timestamps [1, 5], which is a landmark study in computer clock synchronization, is a simple algorithm used to determine the order of events in a distributed computer system. As different nodes or processes might typically not be perfectly synchronized, this algorithm is used to provide a partial ordering of events with minimal overhead, and conceptually provide a starting point for virtual clocks. Lamport's work has an important influence on the clock synchronization in sensor networks since many sensor applications require only relative time, and thus absolute time may not be needed.

The Network Time Protocol (NTP) devised by Mill [17-18] is the most widely used time synchronization scheme in the internet domain. NTP has a lot of advantages such as scalability, self-configuration in large networks, robustness to failures, and ubiquitous deployment. Since NTP is originally designed for synchronizing the computers on the Internet, however, it doesn‘t work well for sensor nodes due to the energy and computation limitations. Global positioning system (GPS) could also be used to synchronize a large group of nodes in the Internet. However, a GPS module may be too expensive to attach on small sensor devices, and its service may not be available in environments such as indoor or under water. Consequently, traditional schemes such as NTP or GPS are not suitable for synchronization method in wireless sensor networks due to the problems such as complexity, energy issues, cost and size factors.

Elson and Estrin [19] have introduced *Post-facto* synchronization scheme in which unlike NTP, each node's clock is normally kept unsynchronized with the rest of the network. That is, a beacon node periodically broadcasts beacon messages to the sensor nodes in its wireless range, and then each sensor node records the time of the event (timestamp) with its own local clock. Upon receiving the reference beacon message, nodes eventually adjust their event timestamps. The local clocks only synchronize when there is difference among the clocks of various nodes. This kind of synchronization is not applicable in all situations, it is limited in scope to the transmission range of the beacon.

The Reference-Broadcast Synchronization (RBS) scheme proposed by Elson *et al*. [20] eliminates the uncertainty of the sender by removing the sender from the critical path. That is, the only uncertainty becomes the propagation and receive time. Unlike the sender-to-receiver synchronization method where the sender transmits the timestamp and the receiver synchronizes, RBS uses receiver-to-receiver synchronization where a third party broadcasts a beacon to all the receivers. The receivers record the time that the packet was received according to their local clocks since the beacon does not contain any timing information. The simplest form of RBS is one broadcast beacon and two receivers. The timing packet is broadcasted to the two receivers. Then, they exchange their timing information and are able to calculate the offset. This scheme can also be extended to a multi-hop scenario. Although this scheme do not consider global synchronization over the entire network, the concept of gateway node is used to extend adjacent nodes synchronization to the synchronization between two nodes that cannot directly communicate with each other.

The Time-Sync Protocol for Sensor Networks (TPSN) [21] is a sender-to-receiver based synchronization scheme that uses a tree to organize the network topology, where the sensor nodes are synchronized to the root node of the hierarchy. Since this scheme is designed as a multi-hop protocol, transmission range is not an issue. The concept is divided into two phases. First, the level discovery phase creates the hierarchical topology of the network in which each node is assigned a level. Only one root node resides on level zero. Second, in the synchronization phase, all *i* level nodes synchronize with *i*-1 level nodes. This makes all nodes synchronized with the root node. Unlike RBS, TPSN has uncertainty in the sender. They attempt to reduce this non-determinism by time stamping packets in the MAC layer. However, both RBS and TPSN protocols still suffer from the uncertainty of MAC layer time-stamping: the jitter in interrupt handling and decoding time.

Flooding Time Synchronization Protocol (FTSP) [26] improves on the disadvantages to TPSN. This scheme effectively reduces all sources of time stamping errors except for the propagation time. The sender contains its time-stamp of the global time in the message at transmission, and then the receiver records its local time when the message is received. Thus, the receiver can estimate the clock offset by using both the sender's transmission time and the reception time. FTSP uses linear regression in order to keep high precision compensation for clock drift. The network structure is mesh topology instead of a tree topology as in TPSN, where the root is elected dynamically and re-elected periodically. FTSP provides multi-hop synchronization where the receiver nodes synchronize their clocks to the root node. That is, the nodes form an ad-hoc structure to transfer the global time from the root to all the nodes, as opposed to a fixed spanning-tree based approach. Thus, FTSP actually saves the initial cost for establishing the tree and is also more robust against node or link failures and dynamic topology changes in WSN.

Su and Akyildiz [28] proposed the time-diffusion synchronization protocol (TDP) for network-wide time synchronization. It actually achieves global synchronization by multi-hop flooding or directed diffusion [22]. The scheme is comprises of several algorithms where there are multiple cycles in the active phase and each cycle has multiple round. Initially, a set of master nodes is elected. That is, the base station starts sending a special timing message to the entire network. Some of the nodes on message receiving side become masters by a leader election procedure. Then, master nodes broadcast a request message containing their current time, and all receivers send back a reply message. Using these round-trip measurements, a master node calculates and broadcasts the average message delay and standard deviation. Receiving nodes record these data for all leaders and then, they turn themselves into so-called diffused leaders and repeat the procedure. Specifically, the master nodes start the time-diffusion procedure involving elected diffused leaders, multi-hop flooding, and iterative weighted averaging of timings from different master nodes. Eventually the average delays and standard deviations are summed up along the path from the masters. The diffusion procedure stops at a given number of hops from the masters. TDP provides synchronization even without external servers. Hence, it handles well node mobility and failures by using a peer evaluation procedure, but it leads to high complexity and its convergence time is also very high.

Asynchronous Diffusion (AD) protocol [29] differs from time-diffusion synchronization [22, 28]. This scheme is optimal and global time synchronization in that it is fully localized and fault-tolerant. In the asynchronous diffusion-based approach, a node can synchronize with neighbors at any time in any order. The algorithm is very simple: each node periodically sends a broadcast message to its neighbors, which reply with a message containing their current time. The receiver averages the received time stamps and broadcasts the average to the neighbors, which adopt this value as their new time. It is assumed that this sequence of operations is atomic, that is, the averaging operations of the nodes must be properly sequenced. This algorithm can also adapt to limited node failure, adverse communication channel, and node mobility. The fault-tolerant diffusion-based protocol goes one step further in assuming the presence of malicious nodes that exhibit Byzantine faults.

Recently, pairwise broadcast synchronization (PBS) was proposed in [36]. This scheme allows sensor to synchronize itself without sending out any packet by overhearing timing messages from two-way message exchange between the neighbors. In a one-hop wireless sensor network where every node becomes an adjacent neighbor to each other, a single PBS message exchange between two nodes would help all nodes to synchronize, thus significantly reducing the communication overhead for clock synchronization. This scheme is also extended to multi-hop wireless sensor network scenarios in [39].

3. Diffusion-based Algorithms

3.1. Generalized Diffusion Algorithm

Traditionally, diffusion-based algorithms mainly have been used for dynamic load balancing in heterogeneous systems [7-8, 12, 14-16]. A dynamic load balancing algorithm normally uses local communication and can rapidly compute new fair data distribution. This leads an unbalanced system to a global “equilibrium” state by exchanging information/workload only between processors owning neighboring subdomains. A generalization of the algorithm proposed by Cybenko [7] was investigated and then the generalized diffusion algorithm was given with direct explicit expression in [15].

In this scheme, it is assumed that the processors may have various relative speeds and each speed is expressed by a positive real number. A processor's workload is considered to be infinitely divisible, and so it can also be represented through a positive real number. Various communication speeds are considered, and during the computations, the communication topology may change between successive load redistribution phases. $G = (V, E)$ is a connected graph whose vertices correspond to the processors and whose edges reflect dependencies between data residing on different processors. Let $V = \{1,..., p\}$ and $E = \{e_1, e_2,..., e_q\}$. l is the vector of the processors' workloads. c is the vector of the processors' speeds. The following generalized diffusion algorithm (GDA) is a generalization of the classical diffusion algorithm (Algorithm 1).

Algorithm 1 Generalized Diffusion Algorithm (GDA) [15]

```
1:  $k$=0;
2:  while not converged do
3:    for all node $n_i$ do
4:      for all neighbors $n_j$ of $n_i$ do
5:        send $l_i^{(k)}$ to $n_j$;
6:        receive $l_j^{(k)}$ from $n_j$;
7:        $\delta_{ij}^{(k)} = m_{ji} l_i^{(k)} - m_{ij} l_j^{(k)}$;
8:        $l_i^{(k+1)} = l_i^{(k)} - \sum_{\{i,j\} \in E(G)} \delta_{ij}^{(k)}$;
9:        $k$=$k$+1;
10:     end
11:   end
12: end
```

In the above algorithm, $l^{(0)}$ is the initial workload vector, $l^{(n)}$ is the workload vector after the nth iteration, and m_{ij} is diffusion parameter, which represents the fact that only a fraction of the difference of load between processor i and its neighbors is sent or received. The algorithm can be expressed in a matrix form as an iterative process of type $l^{(n+1)} = M \cdot l^{(n)}$ where diffusion matrix M is a p x p nonnegative matrix such that $m_{ij} > 0$ iff $\{i, j\} \in E$ or $i = j$, $\sum_{i \in V} m_{ij} = 1$ for all $j \in V$, $m_{ij} c_j = m_{ji} c_i$ for all $i, j \in V$. The diffusion algorithm, as described in Algorithm 1, operates on the load itself. At each iteration of the algorithm, the new load $l_i^{(k+1)}$ of a vertex i is given by the combination of its original load $l_i^{(k)}$ and the load of its neighboring vertices, namely $l_i^{(k+1)} = l_i^{(k)} - \sum_{\{i,j\} \in E(G)} \delta_{ij}^{(k)}$, $i, j \in V$, k = 0, 1, 2, ... This load-balance algorithm fits into the robust interconnection network well. However, further investigation is needed to understand its application to wireless sensor networks, where the communication channel is not perfect, nodes are prone to failure, and the system may be mobile.

The dynamic load balancing algorithms by diffusion [7-8, 12, 14-16] have actually an influence on the time-diffusion synchronization methods in large-scale distributed WSNs [28-29, 40, 46]. In more detail, time-diffusion synchronization protocol (TDP) proposed by Su and Akyildiz [28] is to start from a master node, adjust the clocks of its neighbors, and diffuse this clock adjustment to other nodes. This scheme assumes no specific master nodes and diffusion nodes: Every node is a master node or a diffusion node in a broad sense. This property enhances the robustness of the algorithm. Afterward, Li and Rus [29] also proposed a diffusion-based method for global synchronization in WSNs. This method, which is localized and fully distributed, achieves global synchronization by spreading the local synchronization information to the entire system. They have defined a rate-based diffusion protocol where the synchronization is achieved in the nodes by flooding the information about each node's local clock value. When each node has learned the clock values of all its neighbors, the node can use a mutually agreed consensus value to adjust its clock.

When the above generalized diffusion algorithm is deployed for the diffusion-based synchronization in WSNs, generally, the diffusion synchronization method can be viewed as a high level framework for global synchronization. The low level implementations depend on the way to compute the clock difference among all sensor neighbours. Typically, the diffusion algorithm can choose various global values to synchronize the network according to the consensus that each node in the network agrees to change its clock reading. A simple algorithm for synchronization is to choose the highest or lowest reading value over the network. However, this synchronization is likely to be ruined if there might exist the faulty or malicious nodes with an abnormally high or low clock reading.

3.2. Asynchronous Diffusion Method

Li and Rus actually defined both rate-based synchronous diffusion and asynchronous averaging diffusion protocols for time synchronization in WSNs [29]. The synchronous method assumes all the nodes perform their local operations in a set order, while the asynchronous method relaxes the constraint by allowing each node to perform its operation at random. In these methods, synchronization is done locally without a global synchronization initiator, and can also be done at arbitrary points in time as opposed to the strict timing requirements of the previous synchronization methods.

There are two typical basic operations in diffusion-based synchronization scheme: 1) the neighboring nodes compare their clock readings at a certain time point and 2) the nodes change their clock accordingly. Since the clock comparison and clock update may usually take several steps, however, they both cannot be done simultaneously. This means that the clock updates by using the clock readings at the comparison time will be incorrect. One of the solutions is to use the elapsed time in the clock update. That is, each node keeps a record of how much time elapses after the clock comparison on each node. The complexity of this protocol is also very high as it requires more synchronization rounds to reach reasonable convergence when compared to the previous methods. The convergence speed of the diffusion method is slow compared to that of traditional synchronization algorithms, but it is useful when only a coarse synchronization is required. The diffusion method is actually independent of and thus can be built upon any local synchronization scheme (*e.g.*, averaging clock readings from neighboring nodes). The error in this diffusion method depends on the error inherent to the local synchronization scheme [29].

Actually, the rate-based diffusion algorithm by Li and Rus only considers the time difference between two sensor nodes instead of the absolute clock time value. Hence, it is not required that all the sensors must do this local synchronization at the same time. That is, the exchanged value between sensor n_i and its neighbor n_j is proportional to the time difference between them. In order to remove the constraint that the rate-based synchronous diffusion algorithm requires a set order for all the node operations, we here introduce the asynchronous diffusion algorithm in Algorithm 2, which heavily relies on the previous works [15, 29]. In the asynchronous clock synchronization diffusion algorithm (Algorithm 2), a node can synchronize with its neighbors at any time in any order as long as each node always has the chance to be involved in the execution with nonzero probability. This algorithm gives a very simple clock update operation for a node and its neighbors. Each node runs the asynchronous operations on the fly without knowing what other nodes are doing. That is, each node executes the update operation once for clock diffusion although the order of the operations of all the nodes is randomized.

Algorithm 2 Rate-based Asynchronous Diffusion Algorithm

```
1: for each sensor $n_i$ with uniform probability do
2:     read clock value $c_i$ from $n_i$;
3:     while each neighbor $n_j$ with uniform probability do
4:         read clock value $c_j$ from $n_j$;
5:         write back the new value $c_i + r_{ij}(c_i - c_j)$ to each neighbor $n_j$;
6:     end
7:     write back $c_i - \sum r_{ij}(c_i - c_j)$ to $n_i$;
8: end
```

We here need to give basic definitions and notations for more formal description on Algorithm 2. We first assume that the system consists of *n* sensors, and the sensor devices are connected via wireless communication. Then, the sensor network is simply represented as a deduced graph *G*(*V*, *E*) in which the vertex is the sensor and the edge is the sensor connectivity. If two sensor neighbors are within transmission range, the corresponding vertices n_i and n_j have an edge to connect them. That is, the set of vertices, *V* represents the sensor nodes (*e.g.*, $n_i \in V$ and $n_j \in V$) and the edge relationship $(n_i, n_j) \in E$ presents the sensor connectivity if and only if n_i and n_j are involved in the same round. Then, the clock readings of the *n* sensors in the network at time *t*, can be denoted as $C = (c_1^t, c_2^t, \cdots, c_n^t)^T$ where c_i^t is the clock reading for sensor n_i at time *t*, and *T* presents vector transposition. For simplicity, we can use c_i instead of c_i^t. Whenever the Algorithm 2 is then performed, we can know the following matrix *R* is applied to the clock reading vector *C*. More specifically, we can get the clock value diffusion formula $C^{t+1} = R \cdot C^t$, where the matrix *R* applied on the clock reading vector can be described by:

$$R = \begin{pmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ r_{21} & r_{22} & \cdots & r_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ r_{n1} & r_{n2} & \cdots & r_{nn} \end{pmatrix}.$$

The elements in the matrix *R* are the diffusion rates, called diffusion factor, where normally $r_{ij} = r_{ji}$ and $r_{ij} = 0$ if n_i and n_j are not adjacent, and hence $r_{ii} = 1 - \sum_{j,(i,j)\in E} r_{ij} = 1 - \sum_{j \neq i} r_{ij}$.

If n_i and n_j are within their transmission range, c_i and c_j need to be adjusted under the conservation law in order to search for the convergence value for all the sensors in the system. If c_i and c_j are not equal each other, the diffusion value for convergence is proportional to the difference between c_i and c_j. Thus, the diffusion rate $r_{ij} > 0$ can be chosen randomly provided

$\sum_{j \neq i} r_{ij} \leq 1$. Hence, the sensor n_i loses or gains $r_{ij} \cdot (c_i - c_j)$ to its adjacent sensor n_j and eventually loses or gains a total of $\sum_{j \neq i} r_{ij} \cdot (c_i - c_j)$ to its all neighbors.

We assume that the graph G deduced from the WSNs is strongly connected, so the matrix R is irreducible. Moreover, the matrix R is also symmetric and positive because $r_{ij} = r_{ji} > 0$. Thus, the Algorithm 2 has the flavor of convergence of a Markov chain. We can now have $C^{t+1} = R^{t+1} \cdot C^0$ by applying the above diffusion formula iteratively, where $C^0 = (c_1^0, c_2^0, \cdots, c_n^0)^T$ is the initial clock reading distribution at time 0. We can also expect that this time reading vector becomes $C^S = (c^0, c^0, \cdots, c^0)^T$ after running the algorithm, where $c^0 = \sum_{k=1}^{n} c_k^0 / n$. Hence, all the sensors eventually achieve the stable synchronized clock values C^S. We can also know that C^S is an eigenvector of matrix R with respect to eigenvalue 1. As a consequence, Algorithm 2 might achieve global synchronization in the entire WSN since the time vector $C^{t+1} = R^{t+1} \cdot C^0$ converges to the synchronized clock vector C^S. By using an approach similar to the theorem that Li and Rus have applied in [29], therefore, it can be easily proved that Algorithm 2 converges to the global clock value.

4. Fast-Converged Asynchronous Diffusion Method

4.1. Asynchronous Averaging Diffusion Algorithm

In the asynchronous averaging algorithm (by Li and Rus) [29], all the nodes can perform operations in any order as long as each node is involved in the operations with nonzero probability. The main idea of this algorithm is to average all the clock time readings and set each clock to the average time. It gives a very simple average operation of a node over its neighbors. Each node tries to compute the local average value directly by asking all its neighbors about their values; it then sends out the computed average value to all its neighbors so they can update their values.

In this algorithm, the global average value is used as the ultimate synchronization clock reading in order to make the algorithm more robust and reasonable. That is, a node with high clock reading diffuses that time value to its neighbours and levels down its clock. A node with low clock reading absorbs some of the values from its neighbours and increases its values. After a certain number of rounds of diffusion, some error threshold can be achieved, and thus the clock in each sensor eventually converges to a global average value. Specifically, a round is defined to be the time for each node to finish the average operation in the given algorithm exactly once, so the number of rounds required for the entire network to achieve some error threshold signifies the convergence speed.

In this method, a node may have several neighbors for average operation since the entire network is assumed to be connected. If a node is involved in two or more average operations, therefore, these operations must be sequenced due to the assumption that the average operation is atomic. This algorithm, which has been mathematically proved by Li and Rus [29], can typically adapt to the changing network topology, limited node failure, adverse communication conditions, and node mobility. Fig. 1 shows the example of the randomly selected sensor nodes exactly once in each operation group and the sensor nodes adjusted with most recently received value by atomic operation sequence when they belong to several operation groups.

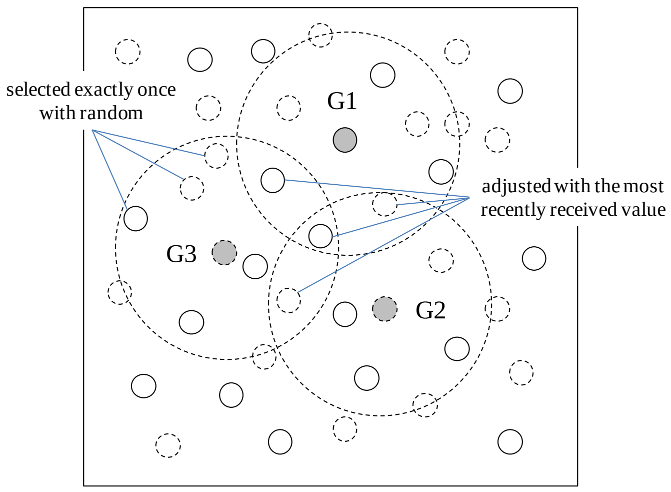
selected exactly once with random
G1
G2
G3
adjusted with the most recently received value

Figure 1. Example of Several Operation Groups

4.2. Fast-Convergence of Diffusion Rounds

In asynchronous averaging diffusion algorithm, each node might have a series of adjusted clock values sent from several neighbors through average operations since all sensor nodes are assumed to be connected. According to the Algorithm of Li and Rus [29] and its assumptions, therefore, a node n_i in each round consequently adjusts its local clock with the most recently received value among a series of average clock values (see Figure 1). Thus, this adjusted clock value is eventually diffused over the whole network. In equation (1), $C_{i\text{-}adjust}$ presents an average clock value sent back to the n_i*'s* neighbors for adjusting clock in the Algorithm of Li and Rus, where N is the number of the neighbors of n_i and C_j is a clock value among neighbor sensor nodes.

$$C_{i\_adjust} = \frac{1}{N}\left(C_i + \sum_{j=1}^{N-1} C_j\right) \qquad (1)$$

We now propose a fast-converged asynchronous diffusion (FAD) scheme (Algorithm 3) for improving the convergence speed of asynchronous averaging diffusion method (Algorithm of Li and Rus) [29]. In Algorithm 3, each node takes the mean of a series of average clock values. More specifically, a node n_i adjusts its clock value at the end of each round with the mean of average clock values sent back from all the neighbors. Since the average operation in this algorithm is not atomic, moreover, the average operations do not have to be sequenced even though a node is involved in two or more operation groups. Hence, the eventual adjusted clock value in a round can be expressed with equation (2), where M is the number of the nodes for average operation within each neighbor group including n_i.

$$C_{i-adjust} = \left(\frac{1}{N}\left(C_i + \frac{1}{M}\sum_{k=1}^{N-1}\left(C_k + \sum_{j=1}^{M-1} C_j\right)\right)\right) \qquad (2)$$

In this algorithm, each node has to keep in the buffer the average clock values sent back from all the neighbors in a round. Then, at the end of a round, every node locally computes the mean with the accumulated average clock values which are already compensated with the offsets according to the time elapsed in the buffer. Algorithm 3 rather seems to increase space complexity and time complexity since it may require more operations and storages than the

Algorithm of Li and Rus in getting average value in each round. However, the number of rounds required for convergence achievement in this scheme is less than that in the asynchronous diffusion method. That is, the overall time for synchronizing all the nodes with this scheme becomes a little shorter since this scheme converges faster than the asynchronous diffusion method. When each sensor node executes the diffusion operation exactly once in each round, it takes $O(n)$ rounds for any clock reading value to propagate to the whole network. Thus, the convergence time is $O(n)$, where n is the number of nodes. In the following sections, the evaluation results of the proposed scheme are presented with NS-2 simulation analysis.

Algorithm 3 Fast-Converged Asynchronous Diffusion Algorithm

```
1: for each node $n_i$ with uniform probability do
2:   read clock values from $n_i$ and its neighbors
3:   average the clock readings
4:   send back to the neighbors the new value (the values are buffered into
       $n_i$ and its neighbors, and accumulated instead of writing over)
5: end
6: each node $n_i$ locally performs average operation again with all adjusted
   buffered values (write the value back to itself $n_i$)
```

Theorem 1. The fast-converged asynchronous diffusion algorithm converges to the stable time clock value (T).

Proof. We prove this result using an approach similar to the theorem proof mechanism of Li and Rus [29]. Let L_t and S_t be the longest value and the shortest value in actually elapsed time clock, respectively, of all sensors in a WSN at time t. That is, L_t is non-increasing over time t since there is no clock value longer than L_t at time t. Similarly, S_t is non-decreasing over time t since L_t cannot decrease from that time by symmetry. From the assumption that L_t is non-increasing, we know L_t is greater than or equal to T. Hence, let I be the infimum of the series L_t, and then we have $\lim_{t\to\infty} L_t = I \geq T$. Now, we suppose $I \neq T$ in order to derive a contradiction.

We consider τ such that $T = I - \frac{n^{n+1}-1}{n-1}\tau = I - \tau(n)$, where the function $\tau(k) = \sum_{i=1}^{k} n^i \tau$ and n is the number of sensors in a WSN (e.g., $\tau(k)$ is the sum of the first k terms of a geometric series with the common ratio $n > 1$). For any k ($k = n, n$-1, ...,1), let A_k be the set of sensors whose clock values are greater than $I - \tau(k)$ and let B_k be the set of the rest of the sensor nodes in a WSN. We know that there must exist a time t such that $L_t < I + \tau$ for some τ and some sensor nodes whose clock values are less than $T = I - \tau(n)$ in a WSN at that time t.

We now consider the inductive proof for kth step ($k = n, n$-1, ...,1) for deriving the contradiction. If the only nodes from A_k (or B_k) are involved in an operation of the proposed algorithm, those nodes must be still in A_k (or B_k) after the operation is completed. However, there must exist an operation that involves the nodes from both sets since WSN is a connected network. Even if at least one node is from A_k and all other nodes have the possible longest elapsed time clock value $I+\tau$, the operation result value is at most $\frac{(I+\tau)(m-1)+I-\tau(k)}{m} < I-\tau(k-1)$, where m is the number of nodes related in this operation. Then, at least $|B_k|+1$ nodes will be in B_{k-1} after that operation is completed. If the nodes from both A_k and B_k are involved in an operation, therefore, these nodes have clock values less than $I-\tau(k-1)$ after the operation is performed.

We know $|B_n| \geq 1$ in operation on the nodes from the sets A_n and B_n at time t since there must be some node whose clock value is less than $T = I-\tau(n)$. After the first operation on the nodes in A_n and B_n is completed, we similarly know $|B_{n-1}| \geq 2$. After the first operation on nodes in A_{n-1} and B_{n-1}, we also know $|B_{n-2}| \geq 3$. Hence, we eventually know $|B_1| = n$ since B_1 is the set of nodes whose clock values are less than $I-\tau(1) = I-n\tau$. This fact contradicts the initial assumption that the infimum of L_t is *I*. Consequently, we have $\lim_{t\to\infty} L_t = T$. In the similar approach, $\lim_{t\to\infty} S_t = T$ can also be easily proved. Therefore, we know that all the clock values on the sensor nodes converge to the stable value *T*.

5. Evaluation Results and Discussions

We implemented the fast-converged asynchronous diffusion (FAD) scheme (Algorithm 3) and asynchronous averaging diffusion scheme (Algorithm of Li and Rus) in simulation with NS-2 simulator (version 2.30) based on IEEE 802.15.4 module. We ran a series of scenarios with different network parameters. In this simulation, the round is defined to be the time for each node to finish the operation in the given algorithm exactly once, so the number of rounds for the entire network to achieve some error threshold signifies the convergence speed. For each experimental set of parameters, the simulation was executed several times using a randomly generated network topology. In each experiment, a stimulus was generated at a randomly chosen node and propagated to the whole network until the relative error (0.01) was achieved. The detail simulation parameters are summarized in Table 1.

Table 1. Simulation Parameters

Parameter	Values
Number of Nodes	75, 90, 100, 125, 150, 200, 300, 400, 500
Sensor Field	100m $\times$ 100m
Transmission Range	15m
Physical Layer & MAC Layer	802.15.4
Routing Protocol	AODV
Relative Error	0.01
Uniform Probability (Mean)	0.5
Threshold (Percentage of Drift)	100%, 80%, 60%, 40%

The number of operation is the number of average operation performed from all nodes in a round. The threshold is the drift rate between the received clock value and local clock value in a tick. In Figure 2 and Figure 3, each data point (*) represents a running on a randomly generated network. That is, the markers are the number of rounds and the number of total operations when relative error becomes 0.01 which is a convergence condition in each experiment. The error rate for convergence decreases exponentially with the increase of the number of rounds [4]. The plotted curve is the average number of rounds and the average number of total operations for one suite of network parameters. A sparse network with fewer nodes undergoes large variation in terms of convergence speed. The simulation results are presented as follows:

Figure 2 evaluates the convergence speed with the number of nodes. The two figures represent the comparison between asynchronous diffusion method (left) and FAD scheme (right) in the number of rounds with threshold value 40%. In this figure, the number of rounds decreases with the increase of the number of nodes. The reason is that the increase of the number of neighbors in each node makes the network more connected and eventually makes the diffusion better expedited. Figure 2 shows that the number of rounds has exponential shape as the number of sensor nodes increases. It means that, when the number of nodes is especially over the specific value, the number of rounds required for achieving global time synchronization are not related with the number of nodes. In this simulation, when the number of sensor nodes is over 150, we can see that the convergence speed of FAD scheme is faster than that of asynchronous diffusion method. More specifically, when the number of nodes increases from 150 to 500, the number of rounds required for achieving global time synchronization in the FAD scheme decreases from 40 to 31.7 while asynchronous diffusion method achieves it in about 38 rounds.

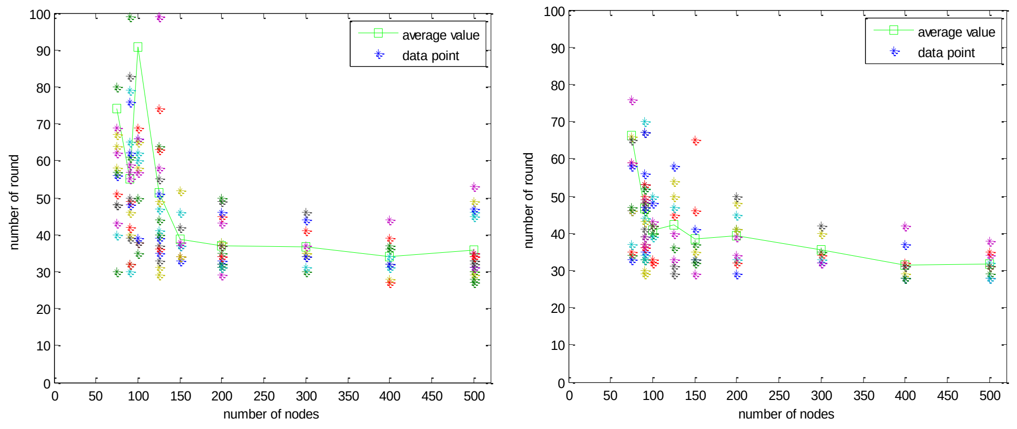

Figure 2. Comparison between asynchronous diffusion (left) and FAD (right) in the number of rounds with threshold value 40%

Figure 3 shows the total number of average operations conducted by all the nodes in each network. The two figures show the comparison between asynchronous diffusion method (left) and FAD scheme (right) in the number of operations with threshold value 40%. These figures represent that the number of total operations increases when the number of nodes increases under the fixed sensor field. The reason is because the number of neighbors increase linearly with the number of nodes with other network parameters fixed. In this simulation, when the number of sensor nodes is 500, we can see that the number of total operations required for achieving global time synchronization in the FAD scheme less than that of the asynchronous diffusion method.

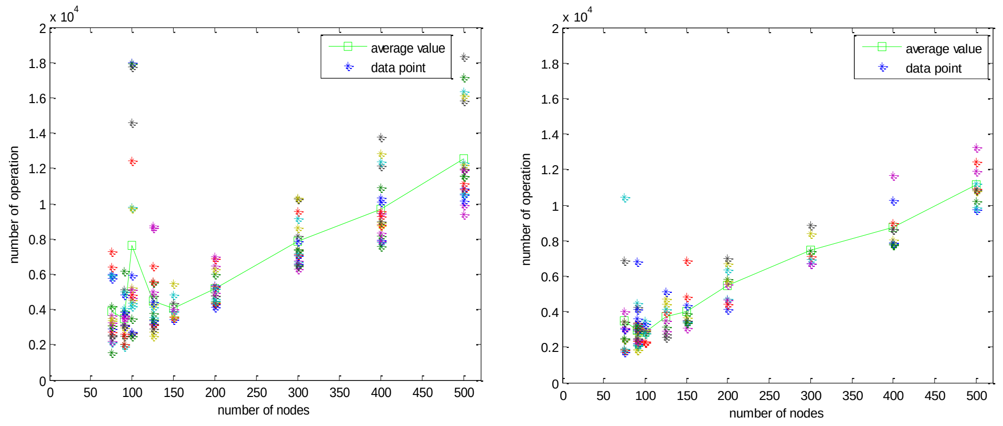

Figure 3. Comparison between Asynchronous Diffusion (left) and FAD (right) with Threshold Value 40% in the Number of Operations

Figure 4 represents the comparison between asynchronous diffusion and FAD in the number of operations and the number of rounds with threshold value (log scale). Figure 4 (left) depicts the number of average operation and Figure 4 (right) shows the number of rounds in this simulation. When the number of nodes is over 175, we can see that FAD requires less operations and fewer rounds than asynchronous diffusion. When the number of nodes is under 175, it might not be possible to compare FAD with asynchronous diffusion from this simulation. When the number of nodes is over 175, however, we can also see that FAD has better performance than asynchronous diffusion.

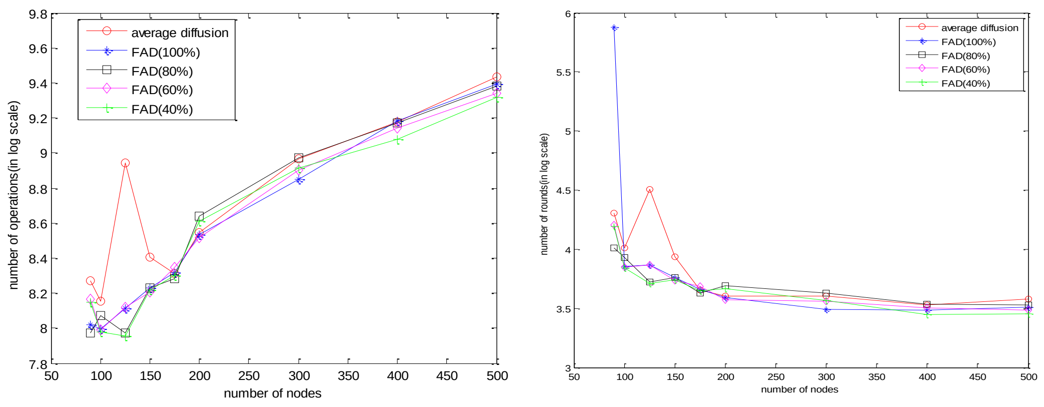

Figure 4. Comparison between Asynchronous Diffusion and FAD in the Number of Operations (left) and the Number of Rounds (right) with Threshold Value (log scale)

6. Conclusion

We consider the diffusion-based algorithms for global clock synchronization in large-scale distributed sensor network. We introduce the rate-based asynchronous diffusion method in which each node can perform its operation at randomly, but still achieve the global clock value over the whole network. We propose a fast-converged asynchronous diffusion synchronization scheme in order to improve the performance of the asynchronous averaging diffusion method, and then prove its convergence mathematically. The evaluation results and discussions for the proposed scheme are also presented with simulation study. We eventually show that the proposed scheme converges a little faster than the asynchronous diffusion method, although it rather seems to require more operations and storages in getting average value in each round.

Acknowledgements

This work was supported by the research program of Dongguk University. Many special thanks are also given to Jiyul Bae for his help on this research work.

References

[1] L. Lamport, "Time, Clocks, and the Ordering of Events in a Distributed System", Communications of the ACM, vol. 21, no. 7, **(1978)**, pp. 558–565.
[2] D. Dolev, J. Halpern and H. R. Strong, "On the Possibility and Impossibility of Achieving Clock Synchronization", Proc. ACM Symp. Theory of Computing (STOC), **(1984)** May.
[3] J. Halpern, B. Simons and R. Strong, "Fault-Tolerant Clock Synchronization", Proc. ACM Symp. Principles of Distributed Computing (PODC), **(1984)** August.
[4] J. Lundelius and N. Lynch, "A New Fault-Tolerant Algorithm for Clock Synchronization", Proc. ACM Symp. Principles of Distributed Computing (PODC), **(1984)** August, pp. 75-88.
[5] L. Lamport and P. M. Melliar-Smith, "Synchronizing Clocks in the Presence of Faults", J. ACM, vol. 32, no. 1, **(1985)** January, pp. 52-78.
[6] H. Kopetz and W. Ochsenreiter, "Clock Synchronization in Distributed Real-Time Systems", IEEE Transactions on Computers, vol. C-36, no. 8, **(1987)** August**,** pp.933–939.
[7] G. Cybenko, "Load Balancing for Distributed Memory Multi-processors", Journal of Parallel and Distributed Computing, vol. 7, no. 2, **(1989),** pp. 279-301.
[8] J. B. Boillat, "Load Balancing and Poisson Equation in a Graph", Concurrency: Practice and Experience, vol. 2, no. 4, **(1990)** December, pp. 289-313.
[9] B. Liskov, "Practical Uses of Synchronized Clocks in Distributed Systems", Distributed Computing, vol. 6, **(1993)**, pp. 211-219, Springer-Verlag.
[10] D. Dolev, J. Y. Halpern, B. Simons and R. Strong, "Dynamic Fault-tolerant Clock Synchronization", Journal of the ACM (JACM), vol. 42, no. 1, **(1995)** January, pp.143 – 185.
[11] S. B. Moon, P. Skelly and D. Towsley, "Estimation and Removal of Clock Skew from Network Delay Measurements", Proceedings of IEEE INFOCOM'99, vol. 1, **(1999)** March, pp.227 – 234.
[12] J. Qiu, K. Miura, H. Inouye, S. Fujiwara, T. Mitsuyu, K. Hirao, Y. F. Hu, and R. J. Blake, "An Improved Diffusion Algorithm for Dynamic Load Balancing", Parallel Computing, vol. 25, no. 4, **(1999)** April**,** pp. 417-444**.**
[13] J. Levine, "Time Synchronization over the Internet using an Adaptive Frequency-locked Loop", IEEE Trans. Ultrasonics, Ferroelectronics and Frequency Control, vol. 46, no. 4, **(1999)** July**,** pp. 888–896.
[14] G. Karagiorgos and N. M. Missirlis, "Accelerated Diffusion Algorithms for Dynamic Load Balancing", Information Processing Letters, vol. 84, no. 2, **(2002)** October**,** pp.61-67.
[15] T. Rotaru and H. -H. Nägeli, "Dynamic Load Balancing by Diffusion in Heterogeneous Systems", Journal of Parallel and Distributed Computing, vol. 64, no. 4, **(2004),** pp.481-497.
[16] P. Berenbrink, T. Friedetzky and Z. Hu, "A New Analytical Method for Parallel, Diffusion-type Load Balancing", Journal of Parallel and Distributed Computing, vol. 69, no. 1, **(2009),** pp.54-61.
[17] D. L. Mills, "Internet Time Synchronization: The Network Time Protocol", IEEE Transactions on Communications, COM 39, no. 10, **(1991)** October**,** pp.1482-1493.
[18] D. L. Mills, "Adaptive Hybrid Clock Discipline Algorithm for the Network Time Protocol", IEEE/ACM Transactions on Networking, vol. 6, no. 5, **(1998)** October**,** pp. 505–514.
[19] J. Elson and D. Estrin, "Time Synchronization for Wireless Sensor Networks", Proceedings of the 2001 International Parallel and Distributed Processing Symposium (IPDPS),Workshop on Parallel and Distributed Computing Issues in Wireless and Mobile Computing, **(2001)** April, San Francisco, USA.
[20] J. Elson, L. Girod and D. Estrin, "Fine-Grained Network Time Synchronization Using Reference Broadcasts", Proc. Fifth Symp. Operating System Design and Implementation (OSDI 2002), **(2002)** December.
[21] S. Ganeriwal, R. Kumar and M. Srivastava, "Time Sync Protocol for Sensor Network", The First ACM Conference on Embedded Networked Sensor System (SenSys), **(2003)** November, pp. 138–149, Los Angeles.

[22] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann and F. Silva, "Directed Diffusion for Wireless Sensor Networking", IEEE Trans. Networking, vol. 11, no. 1, **(2003)** February, pp. 2–16.

[23] K. Romer, "Temporal Message Ordering in Wireless Sensor Networks", IFIP MedHocNet, **(2003)** June, Mahdia, Tunisia.

[24] M. L. Sichitiu and C. Veerarittiphan, "Simple, Accurate Time Synchronization for Wireless Sensor Networks", IEEE Wireless Communications and Networking Conference (WCNC) 2003, vol. 2, **(2003)** March, pp. 1266 – 1273, New Orleans, LA, USA

[25] F. Sivrikaya and B. Yener, "Time Synchronization in Sensor Networks: A Survey", IEEE Network, vol. 18, no. 4, July-August **(2004),** pp. 45-50**.**

[26] M. Maroti, B. Kusy, G. Simon and A. Ledeczi, "The Flooding Time Synchronization Protocol", Proceedings of the ACM Conference on Networked Sensor Systems (SenSys'04), ACM Press, **(2004)**, pp. 39–49, New York.

[27] G. Simon, M. Maroti, A. Ledeczi, G. Balogh, B. Kusy, A. Nadas, G. Pap, J. Sallai and K. Frampton, "Sensor Network-based Counter Sniper System", Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (Sen Sys), ACM Press, **(2004),** New York.

[28] W. Su and I. F. Akyildiz, "Time-Diffusion Synchronization Protocol for Wireless Sensor Networks", IEEE/ACM Transactions on Networking, vol. 13, no. 2, pp.384–397, **(2005)** April.

[29] Q. Li, and D. Rus, "Global Clock Synchronization in Sensor Network", IEEE Transactions on Computer, vol. 55, no. 2, **(2006)** February, pp. 214-226.

[30] A. Giridhar and P. R. Kumar, "Distributed Clock Synchronization over Wireless Networks: Algorithms and Analysis", Proceedings of the 45th IEEE Conference on Decision and Control, **(2006)** December, pp. 4915 – 4920.

[31] O. Simeone and U. Spagnolini, "Distributed Time Synchronization in Wireless Sensor Networks with Coupled Discrete-Time Oscillators", EURASIP Journal on Wireless Communications and Networking, vol. 2007, Article ID 57054, 13 pages, doi:10.1155, **(2007)**

[32] S. Yoon, C. Veerarittiphan and M. L. Sichitiu, "Tiny-sync: Tight Time Synchronization for Wireless Sensor Networks", ACM Transactions on Sensor Networks (TOSN), no. 2, (**2007)** June.

[33] 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", pp.766-777, IEEE Transactions on Communications, vol. 55, no. 4, **(2007)** April.

[34] S. Ganeriwal and C. Pöpper, S. Čapkun and M. Srivastava, "Secure Time Synchronization in Sensor Networks", ACM Transactions on Information and System Security (TISSEC), vol. 11, no. 4, **(2008)** July.

[35] F. Ren, C. Lin and F. Liu, "Self-Correcting Time Synchronization Using Reference Broadcast in Wireless Sensor Network", IEEE Wireless Communications, **(2008)** August, pp.79-85.

[36] K. -L. Noh, E. Serpedin and K. Qaraqe, "A New Approach for Time Synchronization in Wireless Sensor Networks: Pairwise Broadcast Synchronization", IEEE Transactions on Wireless Communications, vol. 7, no. 9, **(2008)** September, pp. 3318-3322.

[37] Q. M. Chaudhari, E. Serpedin and K. Qaraqe, "On Maximum Likelihood Estimation of Clock Offset and Skew in Networks with Exponential Delays", IEEE Transactions on Signal Processing, vol. 56, no. 4, **(2008)** April, pp.1685-1697**.**

[38] P. Sommer and R. Wattenhofer, "Gradient Clock Synchronization in Wireless Sensor Networks", Proceedings of the 2009 International Conference on Information Processing in Sensor Networks, **(2009),** pp. 37-48.

[39] K. -Y. Cheng, K. -S. Lui, Y. -C. Wu and V. Tam, "A Distributed Multihop Time Synchronization Protocol for Wireless Sensor Networks using Pairwise Broadcast Synchronization", IEEE Transactions on Wireless Communications, vol. 8, no. 4, **(2009)** April, pp. 1764-1772.

[40] B. Wang, C. Fu and H. B. Lim, "Layered Diffusion-based Coverage Control in Wireless Sensor Networks", Computer Networks, vol. 53, no. 7, **(2009)** May, pp. 1114-1124.

[41] J. Bae and B. Moon, "Time Synchronization in Wireless Sensor Network," Smart Wireless Sensor Networks, Edited H. D. Chinh and Y. K. Tan, INTECH, **(2010)**, pp. 253-280.

[42] B. -K. Kim, S. -H. Hong, K. Hur and D. -S. Eom, "Energy-Efficient and Rapid Time Synchronization for Wireless Sensor Networks", IEEE Transactions on Consumer Electronics, vol. 56, no. 4, **(2010)** November, pp. 2258-2266.

[43] M. Leng and Y. -C. Wu, "On Clock Synchronization Algorithms for Wireless Sensor Networks under Unknown Delay", IEEE Transactions on Vehicular Technology, vol. 59, no.1, **(2010)** January, pp. 182-190.

[44] J. Chen, Q. Yu, Y. Zhang, H.-H. Chen, and Y. Sun, "Feedback-Based Clock Synchronization in Wireless Sensor Networks: A Control Theoretic Approach", IEEE Transactions on Vehicular Technology, vol. 59, no. 6, **(2010)** July, pp. 2963-2973.

[45] S. M. Rahman and K. El-Khatib, "Secure Time Synchronization for Wireless Sensor Networks Based on Bilinear Pairing Functions", IEEE Transactions on Parallel and Distributed Systems, Digital Object Indentifier 10.1109/TPDS.2010.94, **(2010).**

[46] B. Moon and J. Bae, “A Global Time Synchronization Scheme for Wireless Sensor Networks,” The Proceedings of International Conference - Grid and Distributed Computing (GDC2011), **(2011)** December, pp. 383-391.

[47] Y. -C. Wu, Q. Chaudhari and E. Serpedin, “Clock Synchronization of Wireless Sensor Networks,” IEEE Signal Processing, **(2011)** January, pp. 124-138.

[48] L. Schenato and F. Fiorentin, “Average TimeSynch: A Consensus-based Protocol for Clock Synchronization in Wireless Sensor Networks,” Automatica, vol. 47, no. 9, **(2011)** September, pp. 1878–1886.

Author

Bongkyo Moon, He received the B.S. degree in Computer Science from Sogang University, Korea, in 1992, the M.S. degree in information and communications from GIST (Gwangju Institute of Science and Technology), Korea, in 1998, and the Ph.D. degree in Telecommunications from KCL (King’s College London), London, UK. He worked as a researcher in software and telecommunication areas at INEX Technologies, Inc., Santa Clara, CA, USA from 1992 to 1996 and at ETRI (Electronics and Telecommunications Research Institute), Korea, from 1998 to 1999. He also worked a senior researcher in the Telecommunication R and D Centre, Samsung Electronics, Korea, from 2003 to 2005. Since 2005, he has been working as faculty member (currently associate professor) in dept. of Computer Science and Engineering, Dongguk University, Seoul, Korea. His research interests are mobile computing, cloud computing and network security.