Reactive Clock Synchronization for Wireless Sensor Networks with Asynchronous Wakeup Scheduling

Sang Hoon Lee[1], Yunmook Nah[2] and Lynn Choi[1]

[1] School of Electrical Engineering,
Korea University, Seoul, Korea
{smile97 and lchoi}@korea.ac.kr
[2] Department of Electronics and Computer Engineering
Dankook University, Yongin, Korea
ymnah@dku.edu

Abstract. Most of the existing clock synchronization algorithms for wireless sensor networks can be viewed as proactive clock synchronization since they require nodes to periodically synchronize their clock to a reference node regardless of whether they use time information or not. However, the proactive approach wastes unnecessary energy and bandwidth when nodes don't use time information for their operations. In this paper, we propose a new clock synchronization scheme called Reactive Clock Synchronization (RCS) that can be carried out on demand. The main idea is that a source node initiates a synchronization process in parallel with a data communication. To propagate clock information only when there is traffic, we embed the synchronization process in a data communication process. The results from detailed simulations confirm that RCS consumes only less than 1 percent of the energy consumption compared to two representative existing algorithms while it improves the clock accuracy by up to 75.8%.

Keywords: clock synchronization, wireless sensor network, wakeup scheduling, media access control, preamble sampling

1 Introduction

Time is an indispensable element of information processed by various branches of computer science such as operating systems, distributed systems, and communication networks. Especially, wireless sensor networks (WSNs) that are often used to monitor real-life environmental phenomena require time information to accurately measure external events or to coordinate various operations among the sensor nodes.

Coordinated universal time (UTC) is a time standard based on International Atomic Time (TAI) [1]. UTC is commonly used as a reference time for Internet. However, for wireless sensor networks that often assume GPS-free low-cost sensors,

This work was supported by a Korea University Grant.
This work was supported by Mid-career Researcher Program through NRF grant funded by the MEST (No. 2010-0011271).

a sensor node may not have an access to the UTC. Instead, a node is often designated as a reference node and the reference time offered by this node is used as the standard time to validate the time information. The process of adjusting the clock of each individual node to the clock of a reference node is called clock synchronization.

Most of the existing WSN clock synchronization algorithms [2, 3, 4, 5, 6, 7] require sensor nodes to periodically synchronize their clock to a reference node to maintain the clock error under a certain threshold since they assume that any node can use time information at any time. We can classify these algorithms as proactive algorithms because all the nodes proactively synchronize their clock whether they use time information or not. Since the proactive approach can guarantee a certain level of clock accuracy all the time, it is good for networks where nodes frequently use time information for their operations. Among various operations on a sensor node, MAC operations are the most time sensitive since each node periodically listens to its neighbors to check for a possible communication.

In WSN a node usually employs periodic wakeup and sleep to reduce the energy consumption due to idle listening [8, 9, 10, 11, 12]. The interval and the duration of this wakeup must be scheduled and coordinated with other nodes for an effective communication. Existing wakeup scheduling techniques can be classified into two approaches: synchronous [10, 12] and asynchronous wakeup scheduling [8, 9, 11].

In synchronous wakeup scheduling the wakeup of each node is synchronized with the wakeup of its neighbors. Since nodes share their wakeup schedules and adjust them according to neighbor's wakeup schedule, nodes require a reference time to validate time information for wakeup scheduling. In other words, nodes always need to synchronize their clock to prevent the malfunction of MAC operation. Therefore, the proactive clock synchronization approach is a good choice for the synchronous wakeup scheduling.

In contrast, asynchronous wakeup scheduling allows each node to wake up independently. Since nodes don't share their wakeup schedules, nodes use only duration information to carry out MAC operations. In general, the duration error due to the frequency difference between two clocks isn't big enough to incur a malfunction. For example, the maximum relative error between two Intel PXA271's clocks is 60μs/s [13]. When a node sends a 200-byte packet at 20kbps, the duration error is 0.6μs which is smaller than a single bit transmission time of 50μs. Furthermore, this duration error is not accumulated as time goes by since the duration is valid for an operation. Therefore, the asynchronous MAC may work without the clock synchronization. However, WSNs commonly use time information at an application level to order the chronology of the sensing events or to compute the location information of an event [14]. Thus, sensor networks with the asynchronous wakeup scheduling still require a reference time to validate the time information. However, in WSNs with the asynchronous wakeup scheduling the proactive approach wastes unnecessary energy and bandwidth when there is no event to process.

For the event processing, time information is used by only a source node and a destination node. It means that only the two nodes need to exchange their clock information to synchronize their clocks. Therefore, for WSNs with asynchronous wakeup scheduling we don't need to synchronize the clocks of other nodes who do not participate in the communication.

In this paper, we propose a new clock synchronization scheme called RCS (reactive clock synchronization) that can be carried out in a demand-driven manner. The main idea is that we embed the clock synchronization process in a data communication process so that clock synchronization is performed only when there is traffic. RCS basically uses the offset and delay estimation algorithm [6]. The offset is the difference in the value of a clock from a reference clock. While a pair of nodes exchanges packets for a data communication, each node inserts a timestamp into each packet header. A sender calculates its clock offset to a receiver by using the timestamp. After the calculation, the sender adds the calculated offset to the offset received from the previous hop and delivers the new offset information to the receiver. Then, the receiver can get the accumulated offset from a source node to itself. Therefore, RCS requires three packet transmissions: two for calculating offset and one for delivering offset. By repeating this process hop by hop, the destination node can compute the clock offset from a source node to itself.

To evaluate the accuracy and the energy consumption of the proposed scheme, we perform detailed packet level simulations of RCS and two existing clock synchronization algorithms called TPSN [4] and FTSP [5]. The simulation results confirm that RCS consumes only less than 1% of the energy consumption compared to the existing algorithms when a network has light traffic. In addition, RCS improves the clock accuracy by 75.8% and 36.5% compared to FTSP and TPSN respectively. The reason why RCS can provide more accurate clock than the existing algorithms is that each source starts the clock synchronization right before delivering its message.

This paper is organized as follows. Section 2 introduces the background material for this paper: discussing the existing asynchronous wakeup scheduling algorithms for WSNs. Section 3 presents the proposed reactive clock synchronization algorithm. Section 4 comparatively evaluates the performance of the proposed algorithm using the detailed network simulations. Section 5 surveys the related works. Finally, section 6 concludes the paper.

2 Background: Asynchronous Wakeup Scheduling

In asynchronous wakeup scheduling each node independently wakes up. Since a sender cannot determine when a receiver will wake up, a sender sends a long preamble enough to cover the receiver's wakeup time. On a preamble reception the receiver further wakes up and both the sender and the receiver can participate for the communication. Depending on the packet exchange sequence, we can classify asynchronous wakeup scheduling algorithms into two approaches: a Preamble-Data-ACK (PDA) approach and a Preamble-ACK-Data (PAD) approach.

In the PDA approach, a preamble is a meaningless bit-stream whose only function is to make the receivers of the preamble to prepare for a data packet reception. All the nodes in the transmission range of a sender keep awake after receiving a preamble and prepare to receive a data packet. Only the receiver node which is named in the data packet header keeps receiving the data packet after receiving the header while other nodes go back to sleep. If the receiver successfully receives the data packet, it sends ACK to the sender. B-MAC [11] and WiseMAC [9] are the representative protocols of this approach. In B-MAC a preamble packet is long enough to cover all the neighbors' wakeup time; the minimum length of a preamble should be equal to the duration of wakeup interval. In WiseMAC a receiver node piggybacks its wakeup schedule by using an ACK packet. Therefore, a sender can reduce the length of a preamble by computing the wakeup time of the receiver. Since a receiver inserts the remaining time until the next wakeup into an ACK, WiseMAC does not require clock synchronization for accurate instant time information.

To address the problem due to long preambles, the PAD approach uses multiple short preambles. In addition, by specifying the receiver address in the preamble, all the other receivers of the preamble except the receiver can go back to sleep immediately, avoiding the overhearing problem of unintended receivers in the PDA approach. The receiver node replies to the sender that it can receive a data packet by sending an ACK packet. Then, a sender transmits a data packet without an additional ACK. Since only a receiver keeps awake, other neighbor nodes of a sender can reduce the energy consumption due to overhearing. X-MAC [8] is the representative protocol of this approach. In X-MAC a sender sends a short preamble and waits an ACK packet for a pre-specified duration. If there is no ACK packet, a sender sends additional preamble until it receives an ACK packet.

3 Reactive Clock Synchronization

In this section we introduce Reactive Clock Synchronization (RCS) scheme. After introducing the main idea of RCS, we discuss how RCS can be implemented in two types of the existing asynchronous wakeup scheduling algorithms.

3.1 Main Idea

When a sensor node detects an event, it generates a report data. The data may contain time information such as event detection time. For a destination node to use the time information, the source node and the destination node must share a reference time. Without sharing a reference time which requires a global clock synchronization process, another option is that the destination node converts the time information in the data packet into its local time. If we can compute the offset between the clocks of the source and the destination, the destination node can convert the time information in the data packet provided by the sender to its local time. Intuitively, the offset between a source and a destination is same as the sum of all the offsets between the nodes on a path from a source to a destination. Fig. 1 shows an example of a data delivery. A node *D* has to know the offset from a node *A* to itself to use the time information in the data packet generated by a node *A*. The offset between *A* and *D* (O_{AD}) can be calculated as '$O_{AB} + O_{BC} + O_{CD}$'.

During data communication, each intermediate node accumulates an offset by adding its local offset to the received offset from the previous hop. By using this accumulated offset, a destination node can approximate the offset between a source and itself. Since the offset is calculated on demand during the data communication process, a destination node can use the up-to-date time information. Therefore, we can minimize synchronization error due to clock skew by reducing the gap between the point of the last synchronization time and the point of the time information usage.

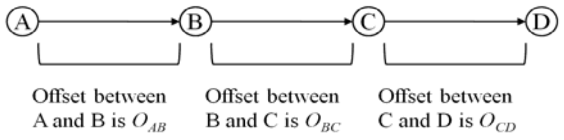

A
B
C
D
Offset between A and B is O_{AB}
Offset between B and C is O_{BC}
Offset between C and D is O_{CD}

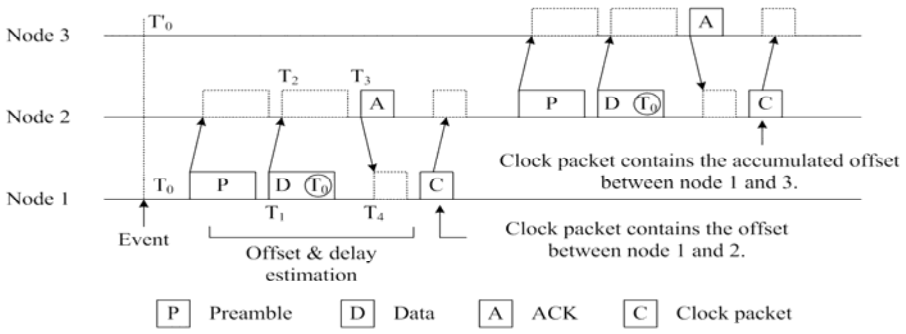
Fig. 1. An example of the accumulated offset

To implement the above distributed clock synchronization algorithm based on the accumulated offset value, we need three-way handshaking: request, reply, and offset delivery. A pair of nodes on a data delivery path carries out the offset-delay estimation algorithm through the request and the reply. After the calculation of an offset a sender delivers its offset value to a receiver. Since each operation requires only timestamps and node address, we can embed the information into the header field of an existing packet if possible.

3.2 In the PDA Approach

In this approach, if a node has a data packet to transmit, it first sends a preamble bit stream which indicates there will be a data packet transmission. We don't use the preamble for the offset-delay estimation since we need two-way message exchange and a sender will send a data packet after sending a preamble. Therefore, we will use a data packet and the following ACK packet to calculate the offset between a sender and a receiver. Since our clock synchronization process requires each node to send the calculated offset, we need an additional packet to deliver the offset. We call this additional packet as *clock packet*. Note that we may not guarantee the reliable transmission for the clock packet. On the clock packet transmission failure the receiver node may reuse the previous clock offset value. If this is the first synchronization between the sender and the receiver, the receiver approximates the clock offset by computing the difference between the send time and the reception time of the data packet, ignoring the transmission delay as in FTSP [5].

Fig. 2 shows an example of the clock synchronization process for the PDA approach where a node 1 is a source and a node 3 is a destination. In this example, to adjust the timestamp in the data packet, node 3 needs the clock offset from node 1. Assume that node 1 detects an event at T_0 and the local time of node 3 is T'_0 at that time. When node 1 sends a data packet, it inserts timestamp T_1 into the packet. Node 2 receives the packet at T_2 and sends an ACK packet at T_3. The ACK packet contains T_2 and T_3. After receiving the ACK at T_4, node 1 can estimate the offset and transmission delay from node 2 and sends a clock packet. Node 2 repeats the same process with the previous communication. After the offset delay estimation, node 2 can compute the offset from node 1 and the offset to node 3. Then, node 2 sends a clock packet which contains the sum of the two offset values. Then, node 3 can get the offset information between a node 1 and itself and adjust the time information in a data packet.



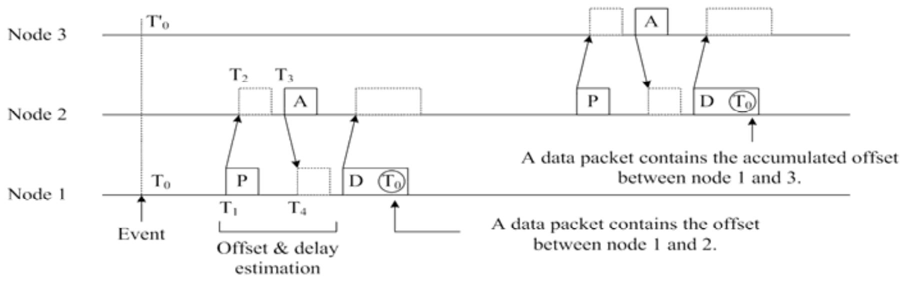
Node 3
Node 2
Node 1
T'0
T0
T1
T2
T3
T4
Event
Offset & delay estimation
P
D
A
C
Clock packet contains the accumulated offset between node 1 and 3.
Clock packet contains the offset between node 1 and 2.
P Preamble
D Data
A ACK
C Clock packet

Fig. 2. An example of an offset estimation process in a PDA scheme

3.3 In the PAD Approach

In this approach we can use a preamble to carry out the offset and delay estimation process since a preamble contains additional information such as the destination address and an ACK packet will follow a preamble. We can carry out the offset-delay estimation while nodes exchange a preamble and an ACK packet. Therefore, we can insert the accumulated offset into the header field of a data packet. Different from the PDA approach, we need no additional packet to deliver the calculated offset.

Fig. 3 shows an example in this approach. Event detection scenario is the same as the previous example in Fig. 2. Although a sender can estimate the offset to a receiver before sending a data packet, it cannot modify the time information in a data packet at a MAC layer since a MAC protocol does not know the content of an application data. Therefore, a sender inserts the offset into the header field of a data packet.



Node 3
Node 2
Node 1
T'0
T0
T1
T2
T3
T4
Event
Offset & delay estimation
P
D
A
A data packet contains the accumulated offset between node 1 and 3.
A data packet contains the offset between node 1 and 2.

Fig. 3. An example of an offset estimation process in a PAD scheme

3.4 Clock Synchronization for Broadcast Data

In WSNs time information can be delivered not only through unicast data but also through broadcast data. For example, a sink can broadcast a query with time information throughout the network. We cannot employ the clock synchronization algorithm introduced in the previous section for a broadcast traffic since ACK packet is usually not required for broadcast data. Thus, we employ the broadcast-based clock synchronization algorithm used in FTSP and S-MAC [12]. The broadcast-based clock synchronization doesn't try to measure the exact offset and delay. Instead, each node simply broadcasts its clock information and it assumes that it can estimate the message delay by using a certain probability function [5] or by assuming a pre-computed (often, zero) message delay [12].

Fig. 4 shows an example of the clock synchronization for a broadcast packet. In this example, node 2 assumes that T_2 is the same as T_1 and a node 3 assumes that T_3 is same as T_4. Therefore, the offset at node 2 and 3 are calculated as (T_2 - T_1) and (T_4 - T_3) respectively.

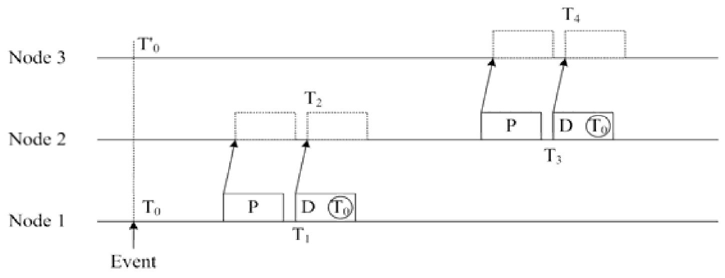


Fig. 4. An example of reactive clock synchronization for broadcast traffic

3.5 Expiration Time

After carrying out the offset-delay estimation, the calculated offset will be useful until the relative clock synchronization error is bigger than the predetermined threshold. If there is burst traffic, multiple data packets will be transmitted for a short time. Therefore, we can reuse the calculated offset for some time. This is called *expiration time*. Each node records the expiration time for every neighbor node after carrying out the offset-delay estimation. If there is another communication before the expiration time, a sender reuses the previous offset information. This can reduce the number of clock synchronization processes when there is burst traffic.

4 Simulation and Result

In this section we evaluate both the energy consumption and the accuracy of RCS by using detailed packet-level simulations. We chose TPSN and FTSP as reference schemes since they are the representative clock synchronization schemes for WSNs. TPSN is based on the offset and delay estimation algorithm while FTSP uses a broadcast-based synchronization. Since the original TPSN doesn't exploit the MAC-layer time-stamping proposed by FTSP, the clock accuracy of FTSP is generally higher than that of TPSN. However, in this simulation we assume the MAC-level time-stamping for both TPSN and RCS for a fair evaluation.

4.1 Simulation Methodology

We have implemented the detailed packet-level simulator using NS-2 [15] to model TPSN, FTSP and RCS. The parameters used in the simulations are summarized in Table 1. We use a grid topology with 400 nodes and a sink locates at the center of the network. We select a random source which generates 10 messages at once, each of which is 50 bytes long. Under each traffic condition, the test is independently carried out 10 times. For this simulation we use B-MAC and X-MAC as underlying MAC protocols. B-MAC is one of the most representative PDA asynchronous MAC protocols while X-MAC is one of the representative PAD MAC protocols.

We use two metrics to analyze the performance of RCS: the average per-node dissipated energy for the synchronization process and the average clock accuracy. The average per-node dissipated energy measures the total energy consumed for each node to carry out the clock synchronization. This metric doesn't include the energy consumption for the idle listening and the data communication. The metric indicates only the overhead required for the clock synchronization. The average clock accuracy measures the difference between the calculated time and the reference time.

Table 1. Parameters for the simulations

Simulation parameters	Value
Cycle time of a node	2 seconds
Time for periodic wakeup	1 ms
Power consumption for the transmission and reception	tx: 30mW, rx: 15mW
Power consumption for the idle state	15mW
Size of control packets used for clock synchronization	10 bytes
Clock synchronization period	10 minutes
Simulation time	1 hour
Number of nodes	400
Number of packets for each message	10
Maximum clock skew	50 ppm

4.2 Average Dissipated Energy

Fig. 5 compares the average per-node energy consumption of each scheme. As shown in Fig. 5 (a) the proactive algorithms consumes much more energy than RCS. RCS consumes only 0.004% energy of TPSN for clock synchronization on a PDA scheme since only those nodes which participate in the data communication carry out the clock synchronization. In contrast, both TPSN and FTSP require every node to participate in the clock synchronization. Therefore, RCS can substantially reduce the energy consumption overhead of the existing clock synchronization algorithms.

Fig. 5 (b) shows that RCS with expiration timer can substantially decrease the energy consumption overhead by suppressing unnecessary clock synchronization. Since a source node sends 10 continuous packets for each message, a source node carries out RCS only once when it sends the first packet of the message. If the time between two messages delivery is less than the expiration time, a source node does not have to carry out RCS when it sends the second message. However, if there is light traffic with small messages, such as a single packet message, the performance gain from the expiration time becomes small. We also find that RCS on the PAD approach consumes 43% more energy than PDA approach since a sender sends multiple short preambles, which results in transmitting duplicated timestamps.

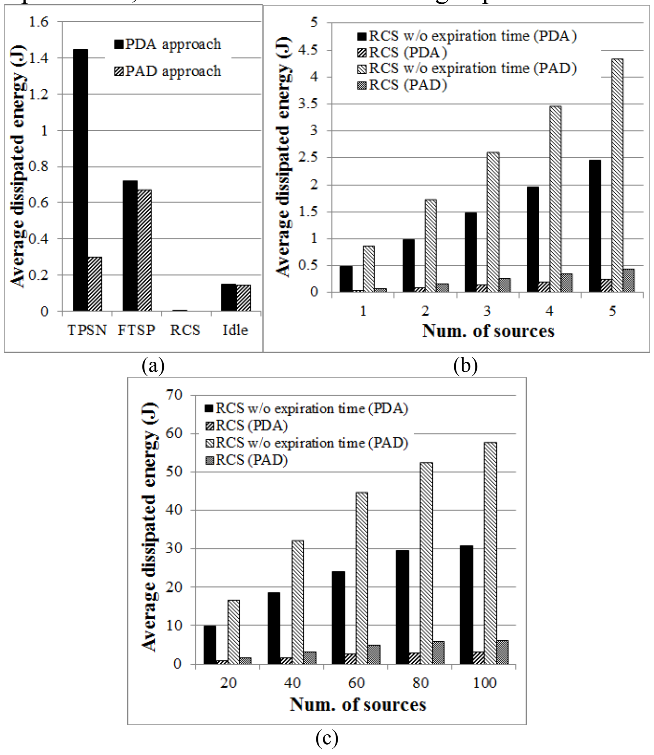


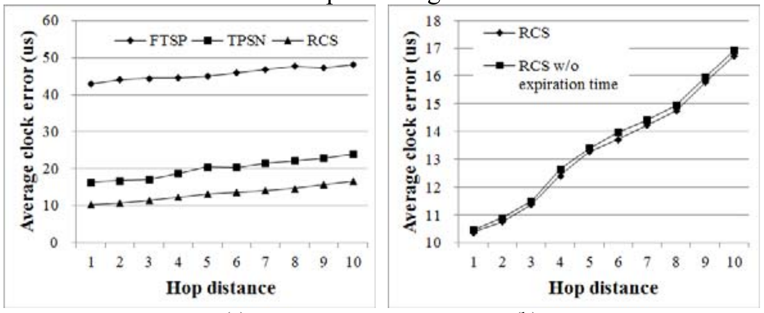
Fig. 5. Average per-node dissipated energy

Fig. 5 (c) also shows that the energy consumption of RCS is proportional to the number of sources. Since the nodes on the path between a source and a destination need to calculate the accumulated offset, the number of nodes which carry out the clock synchronization is proportional to the number of sources. However, as shown in Fig. 5 (b), the rate of energy consumption has slowed as the number of sources increases since different messages can be delivered through the same path. RCS with 100 sources consumes only 3.2 times more energy than RCS with 20 sources. When there are 100 sources, 62.2% of the forwarding nodes which participate in message deliveries can forward two or more messages.

4.3 Average Accuracy

Fig. 6 compares the average clock error of each scheme as we increase the hop distance from a source node. Since FTSP cannot exactly measure the message delay, the average clock error of FTSP is higher than that of TPSN which can compensate the error due to both clock offset and message delay. According to Fig. 8 (a), RCS improves the clock accuracy by 75.8% and 36.5% compared to FTSP and TPSN respectively. The reason why RCS provides a more accurate clock than other schemes is that it carries out the synchronization process right before a destination uses the time information. In contrast to RCS, the existing schemes carry out the process in advance of the use of time information. Therefore, the increased clock skew adds to the clock error as time goes by since the last synchronization.

As shown in Fig. 6 (b), the average accuracy of two versions of RCS has almost the same performance. Since RCS without the expiration time carries out the synchronization process whenever a node sends a data packet, the accuracy of the full version of RCS is slightly higher. However, the expiration time can effectively reduce the energy consumption for the clock synchronization while RCS provides comparable clock accuracy. However, if we average clock error of all the nodes in a network, RCS will have higher total average clock error than the existing algorithms since only the nodes who participate in data delivery carry out clock synchronization process. The total average clock error of RCS including all the nodes in a network was 1.35ms that is 78 times higher than TPSN. Although the total average clock error of RCS is higher than others, it is no consideration since time information of un-synchronized nodes doesn't affect data processing on a sink.



(a) (b)

Fig. 6. Average clock error as we increase the hop distance

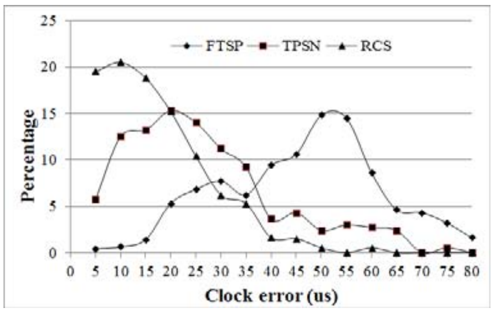


Fig. 7. Distribution of clock error between a pair of nodes

Fig. 7 shows the distribution of clock error from a source to a sink. As expected, the clock error distribution of RCS is closer to zero than others. The maximum clock error of RCS was 58 μs while that of TPSN and FTSP were 72 μs and 77 μs. In RCS 50% of source nodes have clock error under 13 μs while TPSN and FTSP marked 21 μs and 47 μs, respectively.

5 Related Works

Most of the existing clock synchronization schemes proposed for WSNs can be classified as proactive schemes since they perform synchronization process periodically. To improve the accuracy of a clock, most of the existing algorithms often employ the offset-delay estimation algorithm since it can compensate error due to both message delay and clock offset.

Reference Broadcast Synchronization (RBS) [3] tries to improve the accuracy of traditional clock synchronization schemes such as Remote Clock Reading [2] and NTP [6]. In RBS, a reference node broadcasts a reference packet that contains no explicit timestamp. Then, recipients use the packet's arrival time as a reference point for synchronizing their clocks. By comparing the reception time of each receiver, RBS can remove send time and access time from the uncertain message delay factors. However, the message exchanges among receiver nodes cause an exponential increase in the number of synchronization messages. This also increases the energy consumption of RBS exponentially as the number of nodes in the network increases.

To address the scalability issue with RBS, the authors of TPSN [4] aimed at providing a scalable energy-efficient algorithm for WSNs. Similar with NTP, TPSN adopts a two-way message exchange approach to measure the clock offset and message delay. Each node synchronizes with its upper node in the tree hierarchy which is found by flooding a level-discovery packet. However, TPSN does not consider error due to clock skew.

Tiny/Mini-Sync [7] is also based on the two-way message exchange approach. Different from TPSN, this protocol considers the error due to clock skew in addition to the error due to clock offset and delay. To estimate clock skew, Tiny/Mini-Sync uses a history of clock information used in the past synchronization processes. By compensating both clock offset and clock skew, Tiny/Mini-Sync can carry out synchronization more accurately than TPSN.

While conventional synchronization algorithms require a message exchange between a pair of nodes, FTSP [5] adopts a flooding-based approach to further reduce the energy consumption needed for clock synchronization. By using the unidirectional broadcast, FTSP requires only a single clock synchronization message per node instead of a message exchange. In addition, it eliminates timestamp uncertainty by MAC layer time-stamping. However, the flooding scheme of FTSP causes unexpected collision and useless packet transmissions.

6 Conclusion

This paper proposes a new clock synchronization scheme called RCS, which is the first on-demand clock synchronization algorithm. RCS is fundamentally different from the existing clock synchronization schemes in that a source node initiates the clock synchronization process only when there is traffic. This eliminates the overhead for periodic clock synchronization process. In addition, with the expiration timer RCS can further reduce the energy consumption required for the clock synchronization when there is burst traffic. RCS allows us to achieve much lower energy consumption for clock synchronization while preserving a comparable accuracy. The results from our detailed simulations suggest that RCS is very effective in reducing the energy consumption when the network has light or burst traffic, which is the case for WSNs.

References

1. International Bureau of Weights and Measures, http://www.bipm.org/en/home/
2. Cristian, F.: Probabilistic Clock Synchronization. Distributed Computing. vol. 3, pp. 146–158, Springer, 1989.
3. Elson, J., Girod, L., Estrin, D.: Fine-Grained Network Time Synchronization using Reference Broadcasts. In: OSDI 2002, vol 36, pp. 147–163, 2002.
4. Ganeriwal, S., Kumar, R., Srivastava, M. B.: Timing-sync Protocol for Sensor Networks. In: International Conference on Embedded Networked Sensor Systems, pp. 138 ~ 149, 2003.
5. Maróti, M., Kusy, B., Simon, G., Lédeczi, Á.: The Flooding Time Synchronizatoin Protocol. In: SenSys 2004, pp. 39 ~ 49, 2004.
6. Mills, D. L.: Internet Time Synchronization: the Network Time Protocol. Transactions on Communications, vol. 39, No 10, pp. 1482–1493, IEEE, Oct. 1991.
7. Sichitiu, M. L., Veerarittiphan, C.: Simple, Accurate Time Synchronization for Wireless Sensor Networks. In: WCNC 2003, pp. 1266–1273,.
8. Buettner, M., Yee, G. V., Anderson, E., Han, R.: X-MAC: A Short Preamble MAC Protocol for Duty-Cycled Wireless Sensor Networks. In: SenSys 2006, pp. 307–320, 2006.
9. El-Hoiydi, A., Decotignie, J.-D.: WiseMAC: An Ultra Low Power MAC Protocol for Multi-hop Wireless Sensor Networks. In: ALGOSENSORS 2004, LNCS, vol. 3121, pp. 18–31, Springer, 2004.
10. Lu, G., Krishnamachari, B., Raghavendra, C.S.: An Adaptive Energy-Efficient and Low-Latency MAC for Data Gathering in Wireless Sensor Networks. Parallel and Distributed Processing, 2004, pp. 224.
11. Polastre, J., Hill, J., Culler, D.: Versatile Low Power Media Access for Wireless Sensor Networks. In: SenSys 2004, pp. 95–107, 2004.
12. Ye, W., Heidemann, J. S., Estrin, D.: Medium Access Control with Coordinated Adaptive Sleeping for Wireless Sensor Networks. IEEE Transactions on Networking, vol. 12, pp. 493-506, Issue. 3, 2004
13. Intel XScale processor, http://www.intel.com/design/intelxscale/
14. Manley, E. D., Nahas, H. A., Deogun, J. S.: Localization and Tracking in Sensor Systems. In: SUTC 2006, pp. 237-242
15. The Network Simulator ns-2, http://www.isi.edu/nsman/ns