A Performance Comparison of Clock vs. Event Synchronization Protocols

ABSTRACT
Sensor network applications often need accurate temporal information on observations reported from distributed sensor nodes to correctly infer application semantics. Since the nodes' local clocks can go out-of-sync due to clock drifts, a networked time synchronization protocol is needed to keep their clocks synchronized to a reference clock. In this paper, we provide a performance comparison between two classes of time synchronization protocols called clock synchronization (clock-sync) and event synchronization (event-sync), considering different ad-hoc network sizes, node mobility, and traffic volume. The main difference between clock-sync and event-sync is that clock-sync synchronizes nodes' local clocks to a global reference clock, whereas event-sync synchronizes events' generation times from different source nodes to their sink nodes' local clocks. Although these two classes of time synchronization methods have their perspective limitations in application scenarios, they are comparable in that they also share a large application domain with none of these limitations. In order to fully understand the tradeoffs between these two classes of time-synchronization protocols, we have conducted extensive simulations to measure the impact of different network and traffic dynamics on their performance. Our simulations have shown that (1) event-sync provides much better accuracy than clock-sync; and (2) clock-sync scales better in traffic overhead than event-sync under increasing traffic volume. Results of this study are used to derive a selection guideline on how to choose the optimal class of time synchronization protocols under different sensor network and traffic dynamics.

Keywords
Time synchronization, sensor networks, performance evaluation.

1. INTRODUCTION
In recent years, academics and industrial researchers have come up with very creative and successful application scenarios in using wireless sensor networks (WSNs)[1][2] to address a wide array of real-world problems, such as monitoring health conditions of our elders living independently at their home[3][4], tracking endangered species across large remote habitats[5], detecting pollution level in the open ocean, and monitoring soil and pest conditions on agricultural farms[6]. In order to infer correct application semantics, it is necessary to attach accurate temporal information on the observations reported from distributed sensor nodes. Consider a small sensor network with two sensor nodes placed on either side of a room entrance. These two sensor nodes can sense and report the presence of an elder. In order to infer whether an elder is entering or leaving this room, it is necessary to compare the timestamps on two observations reported from two sensor nodes. Since the local clocks on these two sensor nodes can go out-of-sync over time due to clock drifts, comparing their unsynchronized timestamps may lead to an incorrect conclusion. To address this problem of unsynchronized local clocks, a time synchronization algorithm is needed in the wireless sensor network.

The approach taken by traditional time synchronization methods is to synchronize sensor nodes' local clocks to a global reference clock. In this paper, we refer to this class of synchronization mechanisms as clock synchronization (*clock-sync)*. However, not all applications require their nodes' local clocks to be synchronized. For example, if we only need to know the temporal ordering of observations, it is sufficient to synchronize the timestamps among these observations at a sink node that infer application semantics. This class of time synchronization methods is called event synchronization (*event-sync)*.

These two classes of time synchronization have different assumptions and limitations on application scenarios. For examples, the clock-sync does not work in sparse wireless sensor network in which the sensor nodes may not always be fully connected. On the other hand, the event-sync does not provide a global reference clock to the applications. Despite their differences, these two classes of time synchronization protocols share a large domain of sensor network applications, in which the networks are not sparse and the knowledge of relative event time is sufficient. Take the examples of the sensor network applications that track the in/out-flow of merchandizes in stock or monitoring the habitat of a bio-diverse island. There are a limited number of more powerful *sink (or called gateway) nodes* that collect observations from sensor nodes and then execute application logics to process, e.g., temporal information on these observations. In these applications, it is sufficient to synchronize the timestamps contained within *observations* according to the sink node's clock, rather than to synchronize the sensor nodes' clocks with the sink node's clock or a global clock. Under this common application domain, these two classes of time synchronization mechanisms are both applicable; therefore, it becomes meaningful to understand and compare their performance tradeoffs and to help application developers choose the appropriate class of time synchronization under different network and traffic scenarios.

We have not found any existing studies that compare performance of these two classes of time synchronization mechanisms under different network and traffic dynamics (e.g., node size, node mobility, traffic volume, etc.). Without them, developers of sensor network applications can only rely on their intuitions to predict their performance. For example, in a large scale sensor network, intuitions say that since clock-sync maintains a *global clock* by exchanging sync messages (overhead) across a large number of sensor nodes, it is likely to generate a high volume of overhead traffic; thus expensive. Intuitions also suggest that for a traffic pattern of infrequent events, event-sync is likely to produce a smaller amount of overhead because it synchronizes only a small number of events traveling a limited area of the network. Nonetheless, these are only intuitions. Motivated to test these intuitive hypothesis, we compare quantitative the performance tradeoffs between the two classes of time synchronization mechanisms.

We believe that our work is the first to provide detailed and quantitative analysis comparing these two classes of time synchronization mechanisms. We have selected two recently proposed synchronization protocols from each class: TPSN [7] representing the clock-sync class and TSS [8] representing the event-sync class. We have conducted extensive ns-2 simulation experiments on TPSN and TSS to evaluate, analyze, and compare their performances under different network and traffic dynamics. Based on the simulation results, we derive a *selection guideline* on how to choose the better time synchronization mechanisms given different performance requirements and network and traffic dynamics.

The remainder of this paper is organized as follows. Section 2 presents the background of time synchronization for wireless ad hoc networks. Section 3 describes simulation setup and implementation of two classes (clock-sync and event-sync) of mechanisms in the simulator. It also defines evaluation strategy and evaluation metrics for comparing their performance. Section 4 reports our simulation results and performance comparison. Section 5 derives the selection guideline from the simulation results. Section 6 draws our conclusion and future work.

2. BACKGROUND

2.1 Related Work

Time synchronization mechanisms for wireless sensor network can be categorized into two general classes – clock synchronization and event synchronization.

In the clock synchronization, several promising algorithms have been proposed in recent years. For examples, Elson *et al.* has proposed the Reference-Broadcast Synchronization (RBS) [9]. The basic idea of RBS is as follows. In a one-hop neighborhood, a beacon node is selected to periodically broadcast a reference beacon to all its one-hop neighbor nodes. When the neighbor nodes receive this beacon, they exchange their beacon arrival timestamps according to their local clocks. Since all one-hop neighbor nodes are likely to receive the same beacon around the same time, each neighbor node can then estimate the *clock offset* between its local clock and any one of its one-hop neighbor node's local clock by simply taking the difference between its beacon arrival timestamp and its neighbor node's beacon arrival timestamp. To extend this protocol to a multi-hop network, consider a network divided into multiple clusters. There will be some nodes that bridge adjacent clusters (i.e., they are within the intersection regions of two or more adjacent clusters). These bridge nodes can be used to estimate the clock offsets among nodes residing in adjacent clusters. Based on experiments with Berkeley Motes, the RBS authors have reported an average synchronization error of 11 μs (using 30 reference broadcasts) between one-hop neighbors, and the error grows $O(\sqrt{n})$ between nodes that are n hops away.

Moroti *et al.* has proposed the Flooding Time-Synchronization Protocol (FTSP) [10]. The basic idea of FTSP is as follows. A leader node is selected in the sensor network. The leader node's clock is used as the global reference clock. To synchronize other nodes' clocks to the reference clock, the leader node periodically floods the entire sensor network with its current time contained in a sync message. When a node receives a sync message, it records the leader's reference time from the sync message and the arrival time of that sync message. Then it floods this sync message to its one-hop neighbors. Since a node can receive the same sync message multiple times, i.e., one from each of its one-hop neighbors, it can estimate its clock offset and rate difference to the leader node. Based on experiments with the 8x8 grid of Berkeley Motes, the FTSP authors have reported an average synchronization error of 11.7 μs over 10 minutes.

Ganeriwal *et al.* has proposed the Timing-sync Protocol for Sensor Networks (TPSN) [7]. TPSN is based on a spanning tree structure that connects all the nodes in the sensor network. TPSN first selects a node to be the root of this spanning tree. This root node periodically broadcasts a sync-request message to its immediate child nodes in the spanning tree (1st level nodes). After the root node completes pair-wise synchronization with the 1st level nodes, the 2nd round of pair-wise synchronization is started between the 1st level nodes and its immediate child nodes (the 2nd level nodes). The round of pair-wise synchronization continues further down the spanning tree until all nodes are synchronized. Based on experiments with two adjacent Berkeley Motes, the TPSN authors have reported an average synchronization error of 16.9 μs.

These three clock-sync methods have all shown a low average synchronization error. We choose TPSN as the representative of the clock-sync class based on two reasons: (1) TPSN is more recent work, and (2) authors of TPSN claims that TPSN can achieve twice as good precision as RBS based on their reimplementation of RBS. We did not choose FTSP because of its flooding mechanism. In a large sensor network, flooding will generate heavy overhead. In addition, given the similarity between FTSP and TPSN in adjusting the clock, we believe that accuracy between them is similar.

Time-stamp synchronization (TSS) [8] by Romer has suggested that instead of synchronizing every node's clock to a global time, one could obtain the event generation time by estimating and accumulating its hop-by-hop delay. This mechanism can determine the event timing relative to the sink's clock, a function that a clock synchronization mechanism can also provide. We choose Romer's mechanism for the comparison because it is the only event synchronization mechanism we have identified from the literature.

2.2 Mechanism

TPSN [7] has two phases in its process: "level discovery phase" and "synchronization phase". A hierarchical structure with a root node is first created in level discovery phase. In synchronization phase, nodes synchronize their clocks to the root node's clock by using the hierarchical structure constructed in the earlier phase.

a) Level Discovery Phase: This phase of TPSN happens at the beginning, when the network has been setup. The root node assigns itself a level 0 and broadcasts a *level_discovery* packet to start this phase. This level_discovery packet holds the node identity and the level number of the root node. When its neighbors receive this packet, they assign themselves a greater level number than received in the *level_discovery* packet, say level 1. Then they continue to broadcast l*evel_discovery* packets with their own node identity and level number. This process lasts until every node in the network is assigned a level number. Note that once a node is assigned a level, it will ignore any other *level_discovery* packets that are received afterward. This makes sure no flooding will congest the network. At the end of this phase, a hierarchical structure with a root node is created for use in the later phase.

b) Synchronization Phase: In this phase, pair-wise synchronization is achieved across the edges of the hierarchical structure built in the earlier phase. We first consider how to synchronize a pair of nodes through a two-way message exchange. In Figure 1, there are two nodes called A and B. t_1 and t_4 are the time measured according to node A's local clock; t_2 and t_3 are the time measured according to node B's clock. At time t_1, node A sends a *synchronization_pulse* packet to node B. The *synchronization_pulse* packet holds the level number of node A and the value of t_1. Node B receives this packet at t_2, where t_2 is equal to $t_1 + \Delta + d$. Δ represents the clock drift between the two nodes, and d represents the propagation delay. At time t_3, node B sends back an *acknowledgement* packet to node A. This *acknowledgement* packet holds the level number of node B and the values of t_1, t_2, and t_3. Node A receives the packet at t_4. Assuming that the clock drift and the propagation delay do not change in this small period of time, node A can calculate the clock drift and propagation delay using (1):

$$\Delta = \frac{(t_2 - t_1) - (t_4 - t_3)}{2}; d = \frac{(t_2 - t_1) + (t_4 - t_3)}{2} \quad (1)$$

Node A can then synchronize its local clock to B's by having information about the clock drift between them.

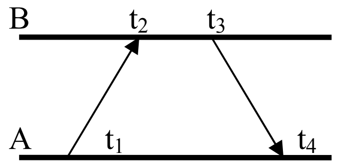

Figure 1. It shows the pair-wise synchronization of TPSN, where t_2 and t_3 are measured in Node B's clock, and t_1 and t_4 are measured in Node A's clock.

The root node starts this phase by broadcasting a *time_sync* packet. Upon its reception, the nodes in level 1 wait for a random time then send a *synchronization_pulse* packet to the root node. The randomized waiting prevents collisions caused by the contention for media access. The root node replies *acknowledgement* packets accordingly. Therefore, all nodes belonging to level 1 can correct their clocks according to the clock of the root node. In addition, the nodes in level 2 will overhear the two-way message exchange because they have at least a neighbor at level 1. Consequently, the nodes in level 2 will send *synchronization_pulse* packet to their level 1 neighbors for synchronization. This is applied recursively with nodes in level *i* synchronizing their clocks to nodes in level *i-1*. Eventually, every node in the network has its clock synchronized to the root node, and the global clock synchronization is achieved.

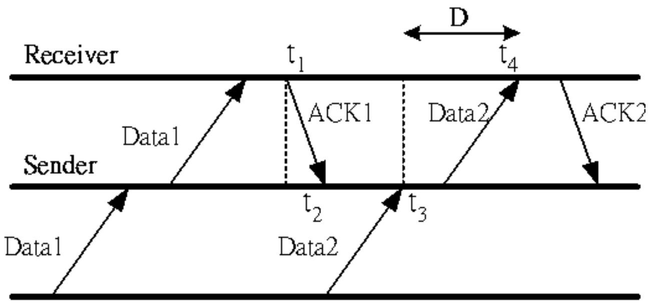

Figure 2. It illustrates the event-sync of TSS. ACK1 departs at *t1* and arrives at *t2*. Data2 arrives at the sender at *t3*, and arrives at the receiver at *t4*. *D* is the hop latency of Data2 at the sender.

Instead of synchronizing every node's clock, event-sync estimates and accumulates the hop-by-hop latency. When a data packet arrives at the sink, one can trace back, from the accumulated latency, the packet generation time relative to the sink's clock. This mechanism can determine the relative data generation time, a function that a global clock synchronization mechanism provides as well.

The event synchronization approach is first proposed by Romer [8]. The assumption is that the wireless links among the nodes employ a CSMA/CA-like MAC-layer mechanism where an acknowledgement is sent per data to assure the reception of the data packet. The hop latency, *D*, can be estimated using (2).

$$D = (t_4 - t_1) - (t_3 - t_2) - (t_2 - t_1) \quad (2)$$

As depicted in Figure 2, t_4 - t_1 can be obtained using the receiver's clock and $t_3 - t_2$ from the sender's clock. The value of $t_3 - t_2$ can be piggybacked on the Data2 packet to the receiver. With the approximation of $t_2 - t_1$, which is the sum of the transmission and propagation delay of ACK1, one may calculate the hop latency *D* of Data2 at the receiver node. Immediately, the latency can be accumulated and carried along with the data packet. Note that to be able to estimate the hop latency with equation (2), each node needs to keep two extra states: the ACK departure time and the ACK arrival time of the latest data packet. In addition, the mechanism will not be able to estimate the hop latency for the first data packet of a flow.

3. SIMULATION

We have implemented the two time synchronization mechanisms on the ns-2 simulator [11]. With an extensive set of simulations, we compare the two mechanisms in terms of accuracy and overhead. Below, we provide the specifics of the simulation setup, evaluation metrics, and evaluation variables.

3.1 Simulation Setup

In all of our simulations, we place the sensor nodes on a predefined grid in a uniformly random fashion. The data sink is fixed in the center of the grid with data sources randomly chosen from others nodes. The communication range of all nodes is set to be 250m. Other simulation setup options include directed diffusion, a well-known data-centric routing mechanism, and IEEE 802.11, a popular wireless link technology. The simulation time is 400 seconds. The data accounted are restricted to those collected after 100 seconds simulation time. This avoids taking the start-up time instability into simulation results.

For all of the evaluation parameters, the base case is defined to have 50 nodes on an 1118x1118m^2 grid, and 10 of the sensor nodes are data sources. Each source sends a 100 bytes data packet every 5 seconds. Unless specified otherwise, these are the default values for the parameters.

3.2 Evaluation Metrics

In order to evaluate the performance of event synchronization and clock synchronization, the following two metrics are investigated:

a) Error: which represents the difference between actual data generation time and estimated data generation time. The correctness of estimated data generation time is important in that it is used to infer temporal relation and ordering of detected events. Inaccurate temporal information can cause incorrect application semantics.

b) Overhead: which represents the portion of traffic produced due to synchronization mechanism over the total traffic. Lower synchronization overhead implies higher throughput and efficiency of the network.

3.3 Evaluation Variables

To compare the error and overhead of the two time synchronization mechanisms, we simulate scenarios with varying network size, node mobility, and data rates.

a) Network Size: For the simulations varying the network size, we change the number of nodes from 20 to 100 with an incremental interval of 10 nodes. In order to fix the network density with increasing number of nodes, we vary the grid size accordingly.

The event synchronization scheme calculates the event generation time by subtracting accumulated hop-by-hop delay from arrival time at the sink node. Therefore, the error tends to grow as the network size increases because the average path length in a larger network tends to be longer. However, the clock synchronization scheme in this study synchronizes the node clocks in a hierarchical fashion. We expect the error of the clock synchronization scheme to grow and in proportion to the rank of the hierarchy that is built for synchronizing the nodes' clocks.

b) Node Mobility: The node mobility model works as follows. Each node has a randomly generated target location and moves to that location with a random speed (maximum speed 20m/s). To change the level of node mobility, we set the pause time between the target locations from 0 to 400 seconds. A smaller pause time implies higher mobility.

In the presence of node mobility, the hierarchy built for clock synchronization may become invalid when a node moves away from its parent node. Therefore, it may need to reconstruct the hierarchy and add to the overhead traffic. Since event-sync uses localized information, i.e., hop delay, it is more adaptive to the change of network topology. Thus, we expect that event-sync simulations experience a lower degree of error than the clock synchronization in the presence of node mobility.

c) Data Rate: For the simulations varying the data rate, we decrease the period of sending a data packet from every 5 seconds to 0.5 seconds, with a decremental interval of 0.5 seconds. Changing the data rate changes the traffic volume.

Since clock synchronization exchanges the synchronization packets at a fixed period independent of traffic volume, the overhead ratio will decrease as the data rate grows, and decrease no more when the network is saturated. Since event synchronization mechanism piggybacks a 16-byte hop-by-hop delay in each data packet, the overhead ratio is a fixed 16% given a packet size of 100 bytes.

4. EXPERIMENTAL RESULTS

We present here the simulation results of the two time synchronization mechanisms. For each simulation scenario, we generate ten random cases. The error and overhead are obtained by running the mechanisms on the same ten random cases per scenario. Each data point on the plots shown here is the average of the ten random cases for each scenario. While the ten random cases appear to be sufficient to indicate the general performance trend, we intend to increase the number of random cases to simulate in the near future.

The accuracy of the clock synchronization mechanism, TPSN, depends on the frequency of the pair-wise synchronization (Section 2.2.B, Synchronization Phase). The more frequent the pair-wise synchronization is done, the higher the accuracy and overhead ratio we observe. In other words, error and overheads are tradeoffs in TPSN. Given that the event synchronization mechanism might not be able to estimate the time for all event packets, it is not fair to compare the accuracy of the two mechanisms by looking at the average error of the valid packets. We, therefore, tune the frequency of the pair-wise synchronization in TPSN such that the overhead ratios of the clock and event synchronization mechanisms are comparable. Hence, in discussing the performance in terms of error, we concentrate on the scaling properties of the mechanisms rather independently. And we compare closely the scaling trend of the mechanisms in terms of overhead.

4.1 Summary of Simulation Results

Figure 3 and Figure 4 depict the error (unit: second) of the clock and event synchronization mechanisms varying the network size, node mobility, and data rate. Figure 6 compares the overhead of the two mechanisms. We find that the error experienced by the event synchronization is significantly lower than that of the clock synchronization in all cases. However, there is an increasing amount of events whose generation time could not be estimated by the event synchronization mechanism when the nodes are mobile. In terms of overhead, clock synchronization scales better when the data rate increases but worse when the network is more dynamic.

4.2 Error details

Figure 3 highlights the error of estimated generation time to real event generation time of clock synchronization under different network sizes, data rates, and node mobility. We observe that the error of clock synchronization mechanism increases generally to the network size, mobility, and data rate. In particular, the error of clock synchronization mechanism is more sensitive to the network size and mobility.

A larger network size implies a higher rank of the clock synchronization hierarchy (Section 2.2.a, Level Discovery Phase). This also increases the accumulated errors along the synchronization hierarchy due to the increased rank, and thus explains the growing trend.

Mobility results in the change of synchronization hierarchy. Until the hierarchy is re-discovered by the mobile node, the clock could continue to go out of synchronization. This explains the increasing error when the network is more dynamic.

TPSN assumes symmetric delays as node pairs synchronize. Given our simulation design where there is only one event sink, the event packets will travel the network towards a particular location. When the data rate is significantly high, we observe a relatively moderate increase in error as the two-way delays become more asymmetric.

Figure 4 shows the errors of event synchronization under different network sizes, data rates, and node mobility. We observe similar scaling trends of the event synchronization mechanism to the clock synchronization mechanism. The

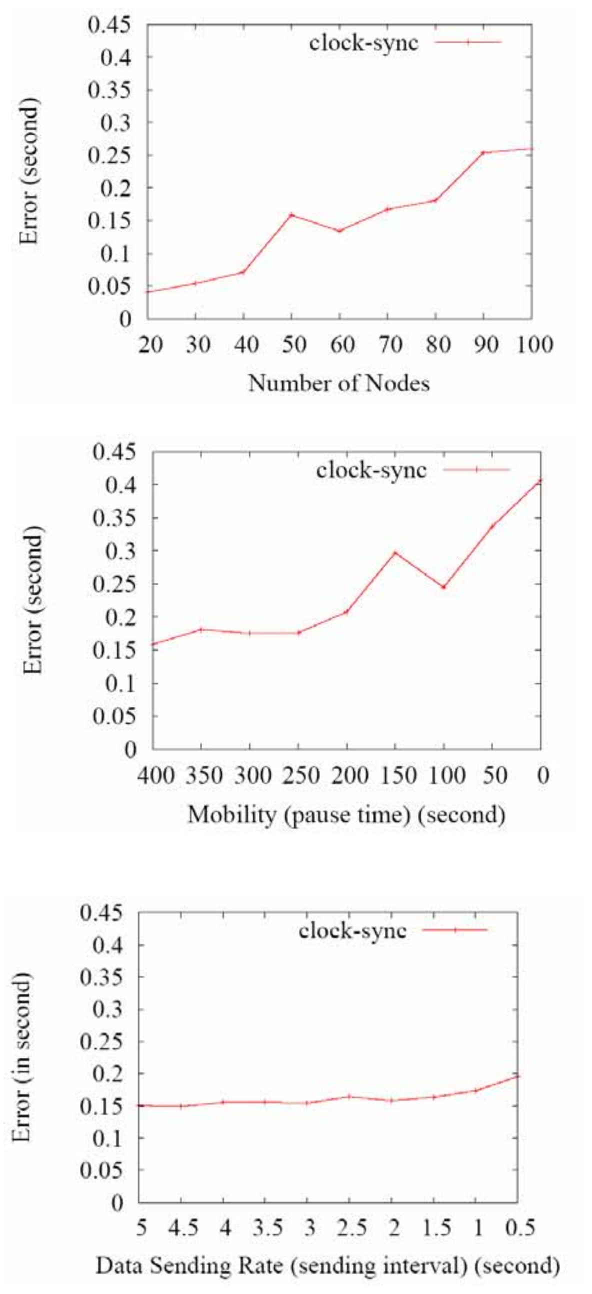

Figure 3. Error due to Clock Synchronization

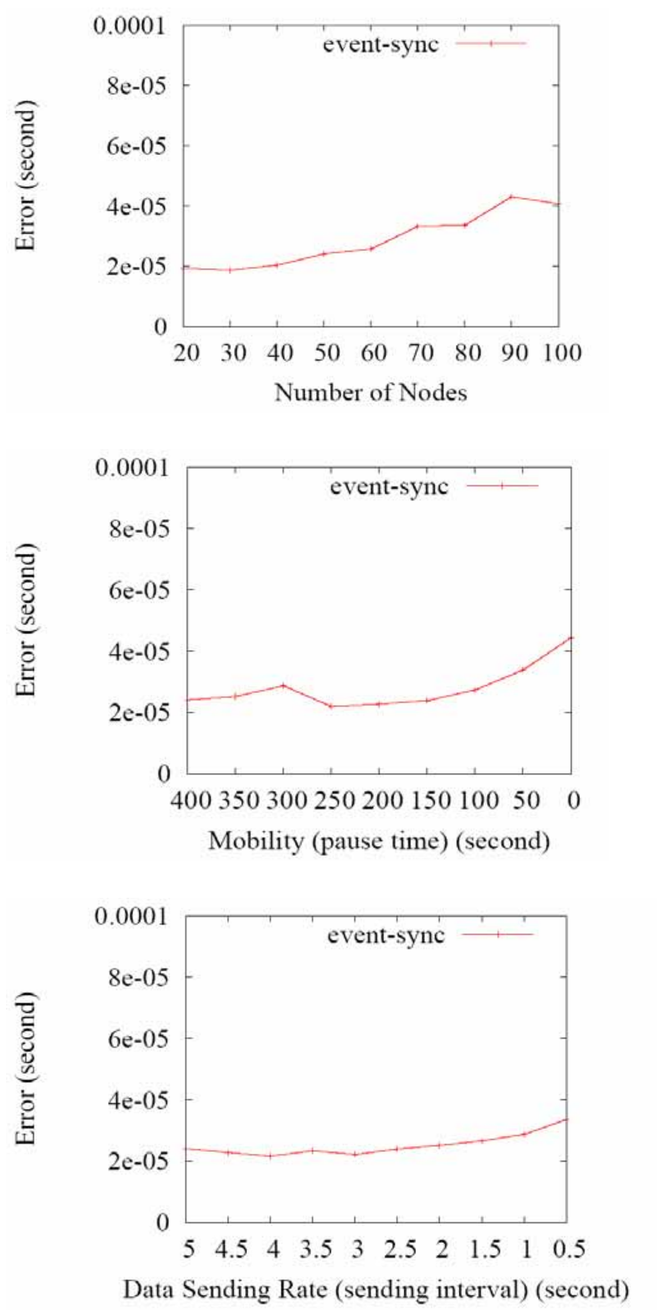

Figure 4. Error due to Event Synchronization.

error of event synchronization mechanism is also more sensitive to the network size and mobility.

A larger network size implies a higher average path length. The error accumulates along the event path from these hop-by-hop latency estimations. The more hops there are in the path, the larger the error.

Mobility influences directly the accuracy of the hop-by-hop latency estimation, especially the part estimating the transmission and propagation delay of ACK. This transmission delay is calculated by dividing the average ACK size to the wireless link bandwidth and the propagation delay is set to the time to reach a node that is about the average distance between 2 nodes apart. When the nodes are mobile, the hop-by-hop latency estimation will be less accurate.

The most significant observation is that the measurable error of event synchronization is much smaller than that clock synchronization, and the difference is about three to four orders of magnitude. We, however, are cautious at concluding whether the event synchronization mechanism is better than the clock synchronization mechanism. The reason is primarily that, using the event synchronization mechanism, we might not be able to estimate the generation time for the first event of a flow.

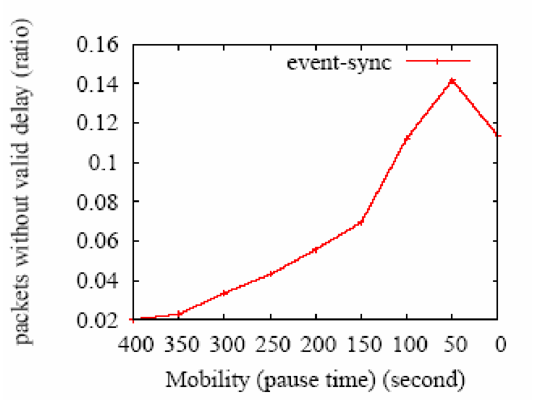

Figure 5. Ratio of packets not synchronized by the event synchronization mechanism.

Consider a dynamic network. When the nodes move away from each other, new paths will be established. The first packets after such path changes, although will help the subsequent estimations of the hop-by-hop latency (Section 2.2, Event Synchronization), will not have sufficient information to compute their own hop-by-hop latencies. As shown in Figure 5, when the mobility is high, the event synchronization mechanism might not be able to estimate the relative generation time for up to 14% of the events sent in total.

4.3 Overhead details

Figure 6 presents the overhead ratio for clock synchronization and event synchronization under different network sizes, data rates, and node mobility. The overhead ratio of event synchronization is constant 0.16, because the overhead of the event synchronization mechanism is a fixed 16-byte field per event packet and the event packets are 100 bytes large on average.

We can see that the overhead of clock synchronization is not very sensitive to the network size. Because when network size increases, the amount of synchronization packets increases and so is the amount of data packets.

We find that the overhead of clock synchronization grows as node mobility increases. This is because the clock synchronization mechanism needs to reconstruct the synchronization hierarchy when the network topology changes. The synchronization hierarchy is reconstructed by sending extra control packets to rediscover the new hierarchy. Since higher node mobility results in higher frequency of topology changes, we observe the growing trend in overhead. In contrast, no such hierarchy rebuilt is necessary for the event synchronization mechanism.

As projected in the Simulation section, the overhead of the clock synchronization mechanism decreases as the data rate increases. We can see that the clock synchronization mechanism scales better than the event synchronization mechanism under high data rates.

5. DISCUSSION

Based on the experimental results, we derive the following

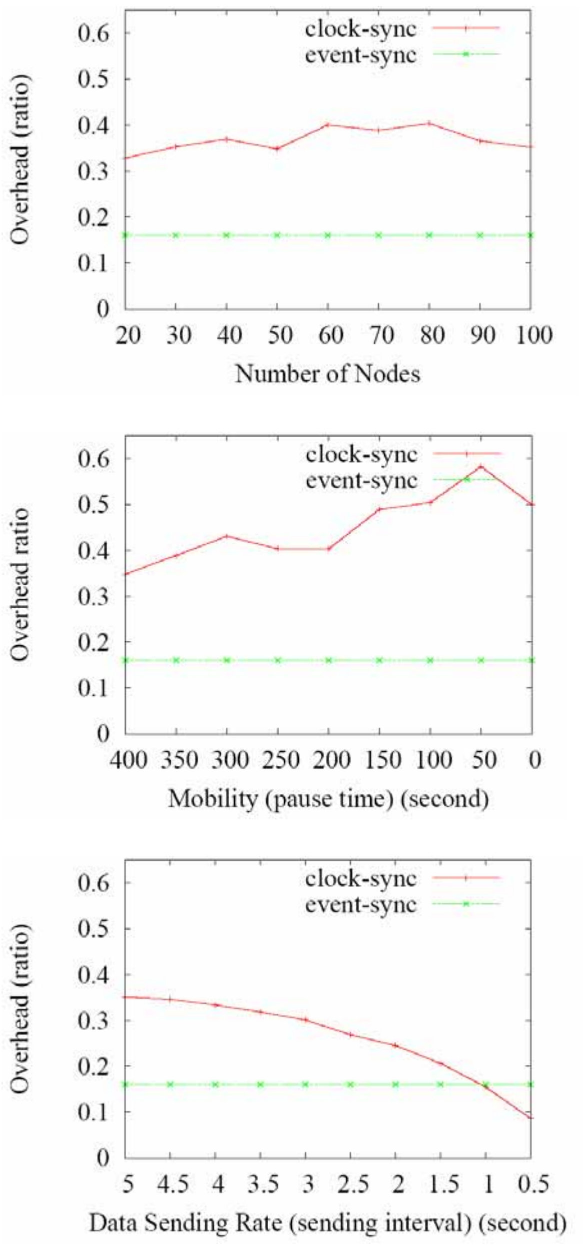

Figure 6. Overhead Comparison of Clock vs. Event Synchronization.

selection guideline for choosing a suitable time synchronization protocol under various ad hoc network and traffic dynamics, as well as application requirements.

(1) When energy consumption is a critical issue and the traffic volume is high, clock synchronization protocol is the better choice, because it involves low overhead to synchronize under high traffic condition.

(2) When node mobility is high, clock synchronization is still superior because all its successfully transmitted packets are labeled with timing information. This is in contrast to event synchronization protocol which has the "broken-path" problem, i.e., a large number of packets can have missing timing labels under high node mobility.

(3) In all other cases, event synchronization protocol is the preferred choice because it has better accuracy than clock synchronization protocol.

Although event sync outperforms clock sync in accuracy under most of the cases, event sync has a fundamental limitation that it only synchronizes the events' generation times to the local clock of the events' sink node. Consider the case that there are more than one sink nodes in the wireless sensor network. Since the clocks of different sink nodes are not synchronized, temporal information from events that go to different sink nodes are not synchronized. If the application requires events to be labeled using a global reference clock, event-sync is not applicable.

In order to solve the limitation of event-sync, we have come up a hybrid approach that combines event-sync with clock-sync. This hybrid method (1) applies event-sync to synchronize all event streams from source nodes to the local clocks of different sink nodes, and (2) runs clock-sync over an overlay network to synchronize local clocks of all sink nodes to a global reference clock. In (2), the pair-wise synchronization procedure between sink nodes in clock sync is now applied to the virtual hop-by-hop link between two sink nodes in an overlay network, which is actually the physical suboptimal shortest path between the two sink nodes in the physical WSN. This hybrid approach combines the advantages of the event-sync (higher accuracy) and clock-sync (global clock). This promising hybrid approach remains to be investigated in the future.

6. CONCLUSION AND FUTURE WORK

With an extensive set of simulations, we conclude the following major findings:

(1) The network size influences strongly the error of the two mechanisms but it has little effect on the overhead ratio.

(2) The mobility impacts both the error and overhead aspects. The event synchronization mechanism begins to experience more events that are not synchronize-able and in the meantime the clock synchronization mechanism begins to degrade in overhead.

(3) The data rate has little effect on the error aspect but it could result in a lower overhead for the clock synchronization mechanism.

The findings in 1 and 2 are rather counter-intuitive whereas the finding 3 is straight forward. From these findings, we are able to derive a set of guidelines for establishing the time synchronization service on the WSNs. When the network size is large, choose the event synchronization mechanism. When the data rate is high, choose the clock synchronization mechanism. When the node mobility is high, choose clock synchronization mechanism if the error is critical or data synchronization mechanism if the overhead is priority.

For a higher degree of confidence, we plan to increase the number of the random cases per scenario in the short-term future. Emerging from the study is a hybrid solution that connects the event sinks using the overlay network technique. The validation and evaluation of the hybrid solution will be for the longer-term future.

7. REFERENCES

[1] I. F. Akyildiz,W. Su, Y. Sankasubramaniam, and E. Cayirci. Wireless Sensor Networks: A Survey. *Computer Networks*, 38:393–422, 2002.

[2] A. Bharathidasan and V.A.S. Ponduru. Sensor Networks: an Overview.

[3] B.G. Celler et al., An instrumentation system for the remote monitoring of changes in functional health status of the elderly, *International Conference IEEE-EMBS*, New York, 1994, pp. 908–909.

[4] G. Coyle et al., Home telecare for the elderly, *Journal of Telemedicine and Telecare,* 1 (1995) 183–184.

[5] A. Cerpa, J. Elson, M. Hamilton, J. Zhao, Habitat monitoring: application driver for wireless communications technology, *ACM SIGCOMM'2000*, Costa Rica, April 2001.

[6] Zhuohui Zhang, Investigation of Wireless Sensor Networks for Precision Agriculture, Paper number 041154, *2004 ASAE Annual Meeting*.

[7] Saurabh Ganeriwal, Ram Kumar, and Mani B. Srivastava. Timing-sync protocol for sensor networks. *In First ACM Conference on Embedded Networked Sensor Systems (SenSys)*, November 2003.

[8] Kay Romer. Time synchronization in ad hoc networks. *In ACM Symposium on Mobile Ad-Hoc Networking and Computing (MobiHoc)*, October 2001.

[9] Jeremy Elson, Lewis Girod and Deborah Estrin, Fine-Grained Network Time Synchronization using Reference Broadcasts, *In the proceedings of the fifth symposium on Operating System Design and Implementation (OSDI 2002),* December 2002.

[10] Miklos Maroti, Branislav Kusy, Gyula Simon, and Akos Ledeczi. The flooding time synchronization protocol. Technical Report ISIS-04-501, Institute for Software Integrated Systems, Vanderbilt University, Nashville Tennessee, 2004.

[11] L. 4 Breslau, D. Estrin, K. Fall, S. Floyd, A. Helmy, J. Heidemann, P. Huang, S. McCanne, K. Varadhan, H. Yu, Y. Xu, and VINT Project. Advances in network simulation. *IEEE Computer*, 33(5):59–67, May 2000.