國 立 台 灣 大 學　碩士學位論文

資訊工程學研究所
指導教授：朱浩華 博士

兩種時間同步化協定的模型建立與比較

Modeling and Comparison of Two Time Synchronization Protocols

研究生：田知本 撰

學　號：R93922045

中華民國九十五年七月

Acknowledgments

First of all, I would like to thank my adviser Prof. Hao-Hua Chu for his advice and patience on me. Prof. Polly Huang provides many valuable insights into the results. Keng-Hao Chang and Tsung-Han Lin contribute great efforts to this work as well.

I would like to also mention my lovable labmates. Chuang-Wen You and Yi-Chao Chen help me a lot on TinyOS programming. Shun-Yuan Yeh, Shin-Jan Wu, and Ko-Chih Wang are my best basketball teammates ever. Ji-Rung Chiang often holds amusements for us. I will never forget these two years with them all.

Finally and the most importantly, I want to express my gratitude to my parents for their support in the past 24 years. It is their love that is the reason I can be here with little accomplishment.

Abstract

To infer correctly application semantics, sensor network applications often need accurate times on observations that are reported from distributed sensor nodes. Since the nodes' local clocks can go out-of-sync due to clock drifts, a networked time synchronization protocol is needed to synchronize their clocks to a reference clock. This paper provides performance modeling and comparison between two time synchronization protocols: TPSN clock synchronization (clock-sync) and TSS event synchronization (event-sync). Their main difference is that the TPSN clock-sync synchronizes all nodes' local clocks to a global reference clock, whereas TSS event-sync synchronizes events' generation times from different local nodes to their sink nodes' clocks. Although these two time synchronization protocols have their respective limitations in application scenarios, they are comparable in that they also share a large domain with none of these limitations. This paper evaluates these two protocols by considering different ad-hoc network sizes, node mobility levels, and traffic volumes. In order to fully understand the tradeoffs between these two time synchronization protocols, we have derived analytical models on their performances and conducted simulations and real experiments to measure the impact of these variables. The experimental results, simulation results, and analytical models all show that (1) event-sync provides much better accuracy than clock-sync, (2) under very high node mobility level, clock-sync may achieve better accuracy than event-sync, and (3) under increasing traffic volume clock-sync scales better. A selection guideline is derived showing how to choose the optimal class of time synchronization protocols under different sensor network dynamics, traffic dynamics, and application requirements.

Contents

List of Figures

Chapter 1

Introduction

1.1 Background

Many recent works have devised very creative and successful applications using Wireless Sensor Networks (WSNs) [3, 4] to address a wide array of real-world problems. These include: monitoring health conditions of the elderly living independently in their homes [8, 9], tracking endangered species across large remote habitats [6], detecting pollution levels in the open ocean, and monitoring soil and pest conditions on farms [15]. In order to infer correctly from the data, accurate times must be attached to the observations reported from distributed sensor nodes.

The traditional approach 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 to a global clock. For example, if we only need to know the relative time of the observations, it is sufficient to synchronize the timestamps among these observations at a *sink (or gateway) node* to correctly infer application semantics. This

class of time synchronization methods is called event synchronization (*event-sync*). The clock and event synchronization mechanisms are different from the event order synchronization mechanisms such as Lamport's logical clock scheme [12] where the finer-grained timing information is no longer available.

These two classes of time synchronization have different assumptions on and limitations for applications. For examples, the clock-sync does not work in a sparse wireless sensor network in which the sensor nodes are not always fully connected. On the other hand, the event-sync does not provide a global reference clock to 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. Consider the examples of sensor network applications that track the in/out-flow of merchandize in stock or monitor the habitat of a bio-diverse island. There are a limited number of more powerful sink nodes that collect observations from sensor nodes. Then, they execute application logics on these observations to process temporal information. 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 to 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. This helps application developers choose the appropriate class of time synchronization under different network and traffic scenarios.

1.2 Motivation and Contribution

No existing studies were found 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. This work is believed to be the first to provide detailed and quantitative analysis comparing these two classes of time synchronization.

Previously, developers of sensor network applications could only rely on their intuitions to predict performance. For example, in a large-scale sensor network, clock-sync maintains a global clock by exchanging sync messages (overhead) across a large number of sensor nodes, hence, generates a high volume of overhead traffic. Intuition suggests this is expensive. Intuition also suggests that for a traffic pattern of infrequent events, event-sync is likely to produce decreased overhead because it synchronizes fewer events and events travel a limited area of the network. Nonetheless, these are only intuitions. To test these intuitive hypotheses, we quantitatively compare the performance tradeoffs.

A recently proposed synchronization protocol from each class was selected: *Timing-sync Protocol for Sensor Network* (TPSN) [10] represents the clock-sync class and *Time-Stamp Synchronization* (TSS) [14] represents the event-sync class. In order to provide a thorough evaluation, analysis, and comparison between their performances and overheads, we conducted the following studies:

- We developed *analytical models* to predict the synchronization accuracy of TPSN and TSS under different network and traffic dynamics.

- We conducted *ns2-simulation* on TPSN and TSS to obtain their accuracy and overheads under different network and traffic dynamics. Also, we showed that our derived analytical models are consistent with simulation results, and can clearly explain them.

- We combine the simulation results and the analytical models to provide a thorough evaluation and comparison between TPSN and TSS. Specifically, the analytical models decompose synchronization error into comparable individual error components. Then we identify and quantify the impact of different network and

traffic dynamic factors on these individual error components.

- We implemented TPSN and TSS on real sensor hardware and provide a realistic comparison of TPSN and TSS on their accuracy and overheads under different network and traffic dynamics.

- Finally, we derive a *selection guideline* showing how to choose the better time synchronization mechanism given different network dynamics, traffic dynamics, and application requirements.

1.3 Thesis Organization

The remainder of this thesis is organized as follows. Chapter 2 presents the background of time synchronization for wireless ad-hoc networks. Chapter 3 describes the protocol mechanisms of TPSN and TSS, and develops analytical models of their accuracy and overheads. Chapter 4 explains the simulation setup and shows the simulation results. Chapter 5 describes the real sensor experimental setup and shows the experimental results. Chapter 6 discusses the selection guideline derived from the simulation results and NTU Taroko results, and draws our conclusion.

Chapter 2

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 were recently proposed. For examples, Elson *et al.* proposed the Reference-Broadcast Synchronization (RBS) [7]. For RBS, within 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 clocks, 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 one-hop clusters. Some nodes bridge adjacent clusters i.e., they are within the intersection regions of two or more adjacent clusters. These bridge nodes are used to estimate the clock offsets among nodes residing in adjacent clusters. Based on experiments with Berkeley Motes, the RBS authors 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.

Maroti *et al.* proposed the Flooding Time-Synchronization Protocol (FTSP) [13]. Basically in FTSP 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 a sync message containing its current time. When a node receives a sync message, it records the leader's reference time and the arrival time. 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 from the leader node. Based on experiments with the 8x8 grid of Berkeley Motes, the FTSP authors reported an average synchronization error of 11.7 μs over 10 minutes.

Ganeriwal *et al.* proposed the Timing-sync Protocol for Sensor Networks (TPSN) [10]. TPSN is based on a spanning tree structure that connects all the nodes in the 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 (first level nodes). After the root node completes pair-wise synchronization with the first level nodes, the second round of pair-wise synchronization begins between the first level nodes and their immediate child nodes (level nodes). The round of pair-wise synchronization continues 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 all show a low average synchronization error. We choose TPSN as the representative of the clock-sync class for two reasons: (1) TPSN is more recent, and (2) the authors of TPSN claim that TPSN can achieve double the precision of RBS. We did not choose FTSP because of its flooding mechanism. In a

large sensor network, flooding generates heavy overhead. In addition, given the similarity between FTSP and TPSN in adjusting the clock, we believe they have similar accuracy.

Time-stamp synchronization (TSS) [14] by Römer suggests 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 determines the event timing relative to the sink's clock, a function that a clock synchronization mechanism can also provide. We choose Römer's mechanism for comparison because it is the only event synchronization mechanism identified in the literature.

Chapter 3

Mechanisms and Analytical Models

We describe protocol mechanisms of TPSN and TSS, and develop their analytical models on performances.

3.1 TPSN (clock-sync) Protocol

3.1.1 TPSN Protocol Mechanism

TPSN [10] 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. Then in synchronization phase, nodes synchronize their clocks to the root node's clock using the hierarchical structure constructed earlier.

1. *Level Discovery Phase*: This phase of TPSN happens at the beginning, after the network has been setup. To start, the root node assigns itself a level 0 and broadcasts a *level_discovery* packet. This 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 packet, say level 1. Then they continue

to broadcast *level_discovery* packets with their own node identity and level number. This process lasts until every node in the network is assigned a level number. Once a node is assigned a level, it ignores any other *level_discovery* packets that are received afterwards. This ensures that flooding does not congest the network. At the end of this phase, a hierarchical structure with a root node is created for use in the next phase.

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

But, exactly how are these synchronization of levels completed? In this phase, pair-wise synchronization is achieved across the edges of the hierarchical structure built in the previous phase. We first consider how to synchronize a pair of nodes through a two-way message exchange. As depicted in Figure 3.1, there are two nodes called N_i and N_{i-1}. t_1 and t_4 are the times measured according to node N_i's local clock; t_2 and t_3 are the times measured according to node N_{i-1}'s clock. At time t_1, node N_i sends a *synchronization_pulse* packet to node N_{i-1}. The *synchronization_pulse* packet holds the level number

of node N_i and the value of t_1. Node N_{i-1} 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 sending delay (including the time to send, propagate, and receive the packet). At time t_3, node N_{i-1} sends back an *acknowledgement* packet to node N_i. This *acknowledgement* packet holds the level number of node N_{i-1} and the values of t_1, t_2, and t_3, and node N_i receives the packet at t_4. TPSN assumes the delays of *synchronization_pulse* packet and *acknowledgement* packet are the same, so t_4 is equal to $(t_3 - \Delta + D)$. Assuming that the clock drift and the propagation delay do not change in this small period of time, node N_i can calculate the clock drift and propagation delay using the following formula:

$$\Delta = \frac{(t_2 - t_1) - (t_4 - t_3)}{2}; D = \frac{(t_2 - t_1) + (t_4 - t_3)}{2} \tag{3.1}$$

Node N_i can therefore synchronize its local clock to N_{i-1}'s since it has information about the clock drift between them.

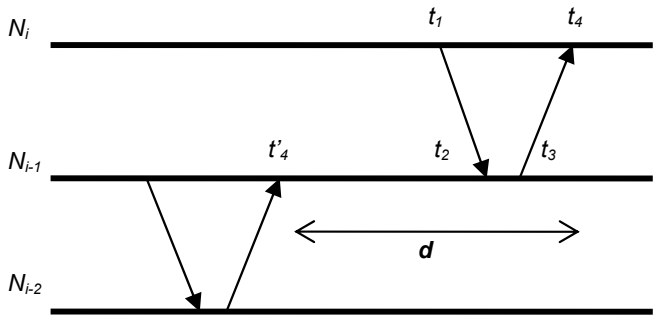

Figure 3.1: Pair-wise synchronization of TPSN. t_2 and t_3 are measured in node N_{i-1}'s clock, and t_1 and t_4 are measured in node N_i's clock

3.1.2 TPSN Analytical Model

We analyze the synchronization error in TPSN, and develop its analytical model. As shown in Figure 3.2, the TPSN synchronization error is composed of three error components: *pair-wise synchronization error* (E_{sync}) caused by the delay estimation when a

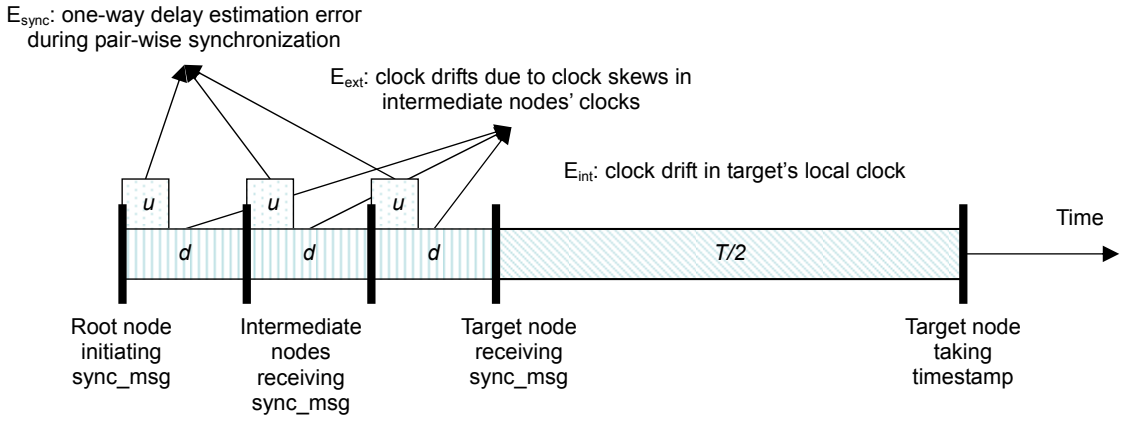

Figure 3.2: TPSN synchronization error decomposed into three error components

parent node is exchanging the global clock value with its one-hop child nodes, *external clock skew error* (E_{ext}) caused by clock skews of intermediate nodes on the transmission path while they are forwarding the synchronization message from the root node and the target node, and *internal clock drift error* (E_{int}) caused by clock skew of the target node as its local clock drifts away from the most recent synchronization time point:

$$E^{tpsn} = E_{sync} + E_{ext} + E_{int} \tag{3.2}$$

The pair-wise synchronization error (E_{sync}) comes from TPSN's assumption that sending delay of *synchronization_pulse* packet and that of *acknowledgement* packet are the same. However, in real deployment, the forward and reverse link delays can be asymmetric. This leads to incorrect calculation of the clock drift value, i.e., Δ in Equation 3.1. Denote the average time difference between the forward link delay and reverse link delay as u. At the end of each pair-wise synchronization (i.e., t_4 in Figure 3.1), the asymmetric link delay will cause the clocks between the parent and child nodes to be off by $\frac{u}{2}$ (on average). Consider a target node at level l, since there are l number of such pair-wise synchronization occurred between pairs of intermediate nodes on the path between the root node and the target node, the total synchronization error is the sum of all pair-wise

synchronization errors on that path. It can be written as follows.

$$E_{sync} = \frac{l \cdot u}{2} \tag{3.3}$$

The external clock skew error (E_{ext}) is caused by clock skews of *intermediate nodes* as the sync message (containing the global clock) is pushed from the root node down the hierarchy to target nodes. Since each pair-wise synchronization takes some amount of processing and transmission time, this hop latency needs to be accounted for by each intermediate node using its local clock, added to the global clock, and then passed it down the hierarchy. This clock skew error is *external* in the sense that the error is not contributed by the target node, but rather clock skews from these external intermediate nodes. Denote the average clock skew (i.e., clock drift rate) on any non-root nodes as r. Denote the average hop latency time for a pair-wise synchronization as d. At the end of each pair-wise synchronization (i.e., t_4 in Equation 3.1), clock skew will cause the clock on an intermediate node to drift apart on average by $(d \cdot r)$ from the global clock. Consider a node on level l, since there are l number of pair-wise synchronization occurred on the path between the root node and the target node, the external clock skew error is the sum of clock skews on that path.

$$E_{ext} = l \cdot d \cdot r \tag{3.4}$$

The internal clock drift error (E_{int}) is caused by the drift of the clock on the target node between the current time and the most recent synchronization time point. This clock drift error is called *internal* in the sense that the error is contributed solely by the target node's local clock. Denote this synchronization time interval as T. If the data generation time occurs uniformly over this time interval, the average amount of clock drift away from the global clock (since the last synchronization time point) can be derived as follows.

$$E_{int} = \frac{r \cdot T}{2} \tag{3.5}$$

The aggregate synchronization error for TPSN (E_{tpsn}) is the sum of these three error components. Combining Equations 3.3, 3.4, and 3.5 gives the following equation for E^{tpsn}.

$$E^{tpsn} = l(\frac{u}{2} + r \cdot d) + \frac{r \cdot T}{2} \tag{3.6}$$

3.2 TSS (event-sync) Protocol

3.2.1 TSS Protocol Mechanism

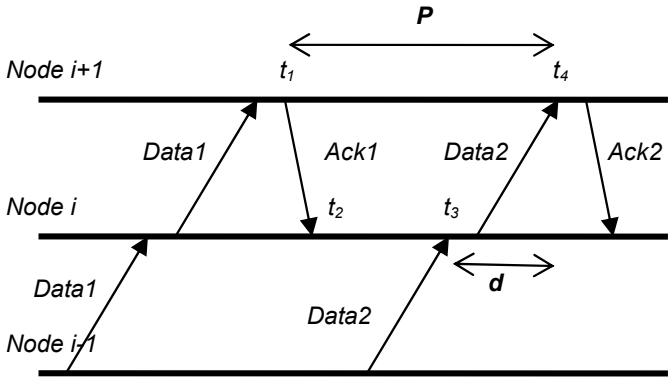

Figure 3.3: Event-synchronization of TSS. ACK1 departs at t_1 and arrives at t_2. Data2 arrives at the sender at t_3, and arrives at the receiver at t_4. d is the hop latency of Data2 at the sender.

Rather than synchronizing every node's clock to a global clock, TSS [14] estimates and accumulates the hop-by-hop latency. When a data packet arrives at the sink, the packet generation time relative to the sink's clock can be traced back from the accumulated end-to-end latency. TSS then determines the relative data generation time to the sink's clock. Wireless links among the nodes are assumed to employ a CSMA/CA-like MAC-layer mechanism where an acknowledgement is sent for each data to assure the reception of the data packet. The hop latency, d, can be estimated using the following

formula.

$$d = (t_4 - t_1) - (t_3 - t_2) - Est_{D_{ACK1}} \tag{3.7}$$

As depicted in Figure 3.3, $(t_4 - t_1)$ 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. $Est_{D_{ACK1}}$ is the estimation of the delay of ACK1. We can use the transmission delay to estimate this value, as in Equation 3.8. This estimation ignores some CPU processing time and the propagation delay.

$$Est_{D_{ACK1}} = \frac{ACK1\_packet\_size}{bandwidth} \tag{3.8}$$

With the above information, the hop latency d of Data2 at the receiver node can be calculated. Immediately, the latency can be accumulated and carried along with the data packet. To estimate the hop latency, each node needs to keep two extra states: the ACK departure time and the ACK arrival time of the latest data packet. When a data packet is ready to be sent but cannot find the state information to be piggybacked, e.g. the first flow of packets, an additional overhead will be sent in order to set up the state information.

3.2.2 TSS Analytical Models

We analyze the synchronization error in TSS and develop its analytical model. The analytical model is similar to that of TPSN, composing of three components shown in Figure 3.4: *pair-wise synchronization error* (E_{sync}) caused by the delay estimation on its transmission path, *external clock skew error* (E_{ext}) caused by clock skews of intermediate nodes while they are forwarding data packets from the source node and the sink node, and *internal clock drift error* (E_{int}) caused by the clock skew of the target node as its local clock drifts between the arrival time of a packet and the arrival time of its subsequent packet carrying its end-to-end delay.

$$E^{tss} = E_{sync} + E_{ext} + E_{int} \tag{3.9}$$

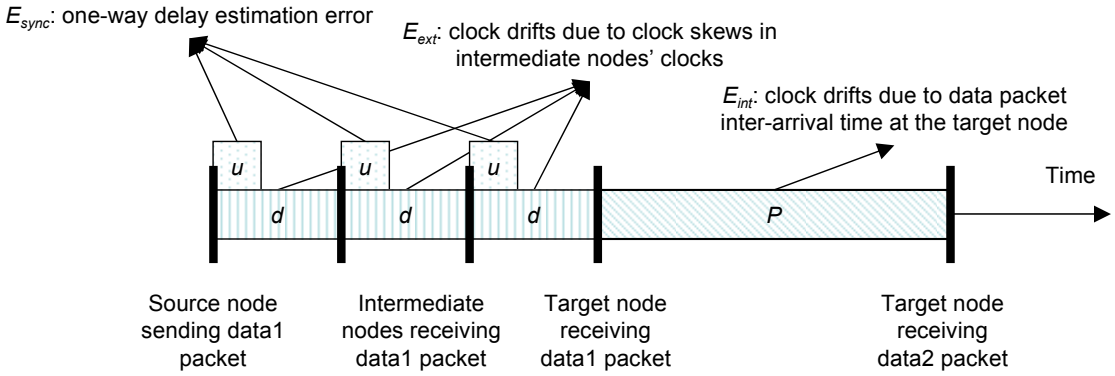

Figure 3.4: TSS synchronization error decomposed into three error components.

The pair-wise synchronization error (E_{sync}) comes from hop delay estimation in TSS. TSS assumes that the ACK packet transmission time ($(t_3 - t_2)$ in Figure 3.3) can be estimated according to Equation 3.8. However, in real deployment, other small delay factors such as the protocol processing time are not considered. As a result, TSS has the delay estimation error. Denote the average delay estimation error as u. Consider a data packet with a path-length of l between a source node and a sink node. Since there are l numbers of such forwarding hops, the total pair-wise synchronization error is the sum of all pair-wise synchronization errors on that path. It can be written as follows.

$$E_{sync} = l \cdot u \tag{3.10}$$

The external clock skew error (E_{ext}) is caused by clock skews of intermediate nodes as the data packet is forwarded from a source node to a sink node, and its per-hop delays are accumulated in the subsequent data packet. Denote the average per-hop delay time as d. At the end of each hop transmission, clock skew will cause the clock of an intermediate node to drift apart on average by $(d \cdot r)$ from the global clock. Consider a data path of length l. The total external clock skew error is the sum of individual clock skew error over these l intermediate nodes. It can be written as follows.

$$E_{ext} = l \cdot d \cdot r \tag{3.11}$$

The internal clock drift error (E_{int}) is caused by the clock drift of the *sink node* between the arrival time of a data packet (e.g., Data2 packet) and the arrival time of the previous data packet (e.g., Data1 packet) carrying the accumulative hop-by-hop delay of the previous packet (e.g., Data1 packet). This clock drift error is internal in the sense that the error is contributed solely by the sink node's local clock. Denote the average inter-arrival time of a packet stream as P. The amount of clock drift from the packet inter-arrival times can be derived as follows.

$$E_{int} = r \cdot P \tag{3.12}$$

The aggregate synchronization error for TSS (E_{tss}) is the sum of these three error components. Combining Equations 3.10, 3.11, and 3.12 gives the following equation for E^{tsss}.

$$E^{tss} = l(u + r \cdot d) + r \cdot P \tag{3.13}$$

3.3 Comparison of Analytical Models of TPSN and TSS

In the analytical models of TPSN and TSS, we have found that they both have three identical error components. The first one is the protocol-specific hop delay estimation error (E_{sync}). In TPSN, this comes from the asymmetry of packet exchange between two nodes. In TSS, this is the propagation delay of an acknowledgement packet.

The second component is the clock skews (E_{ext}) over the end-to-end delay. When the sync message propagates through the network, the time that this information stays on each intermediate node would contribute some errors from the clock skews. In TSS, this is the end-to-end delay of each data packet from a source node to a sink node. In TPSN, it is the end-to-end delay of each sync message from the root node to a target node.

The third component is the amount of local clock drift (E_{int}) over the last synchronization point. In TPSN, synchronization is done periodically, so the amount of local

clock drift is proportional to how fast the network is re-synchronized. Interestingly, this error also shows up in TSS. Because the clocks in TSS are synchronized by the acknowledgement packets, there are also time intervals between the times when the clocks are synchronized and when the local clocks are being used. If the source nodes send data packets in a constant rate, this time interval will basically be the interval of sending data.

Chapter 4

Simulation

TPSN and TSS were implemented on the *ns-2* simulator [5]. We describe the details for the simulation setup, evaluation metrics (error and overhead), and evaluation variables (network size, node mobility level, and traffic volume). Based on the analytical models derived in Chapter 3, we analyze the impacts of changing these evaluation variables on the evaluation metrics. We also verified the analytical model by the simulation results.

4.1 Simulation Setup

In all simulations, the sensor nodes are placed on a predefined grid in a uniformly random fashion. The data sink is fixed in one corner of the grid, while other nodes are randomly chosen as data sources. The communication range of all nodes is set to be 40 meters. Each node has a constant clock skew rate selected from 0.5×10^{-6} to 1.5×10^{-6}. Other setup aspects include directed diffusion [11], a well-known data-centric routing mechanism, and IEEE 802.11, a popular wireless link technology. The simulation time is 400 seconds. The data used are restricted to those collected after 100 seconds simulation time. This avoids taking the start-up time instability into the simulation results.

For all of the evaluation parameters, the base case is defined to have 40 nodes on an $80 \times 80\ m^2$ grid, and 10 of these 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.

An uncertain CPU processing delay before transmitting time-sync messages contributes errors in TPSN and TSS. We however do not to simulate this uncertainty in the simulations for two reasons: (1) the uncertainty is hardware dependent, and there is no model proposed yet to simulate it in a simulator; and (2) omitting CPU processing time is applied to both TPSN and TSS, making it a fair comparison. Omission of CPU processing delay makes the forward and reverse link delay in TPSN perfectly symmetric. In addition, it also reduces the hop-delay estimation error in TSS.

4.2 Evaluation Metrics

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

- *Synchronization Error*: This represents the difference between actual data generation time and estimated data generation time. The correctness of the estimated data generation time is important, because it is used to infer temporal relation and order of detected events. Inaccurate temporal information can cause incorrect application semantics.

- *Overhead*: This represents the traffic produced due to the synchronization mechanisms in proportion to the total data traffic. Lower synchronization overhead implies higher throughput and efficiency of the network.

4.3 Evaluation Variables

To compare the error and overhead of the two time synchronization mechanisms, scenarios are simulated with varying *network sizes*, *node mobility levels*, and *data rates*.

- *Network Sizes*: To vary the network sizes, the number of nodes is changed from 20 to 140 with incremental steps of 20 nodes. In order to fix the network density with increasing number of nodes, the grid size is varied accordingly.

- *Node Mobility Levels*: In the node mobility model, each node has a randomly generated target location and moves to that location with a random speed (maximum speed 10 m/s). To change the levels of node mobility, the pause time between the target locations is adjusted from 0 to 400 seconds. A smaller pause time means higher mobility.

- *Data Rates*: To vary the data rates, we adjust the packet sending rates at the source nodes, from a fixed-size packet every 4 (2^2) seconds to every 0.015625 (2^{-6}) second. A higher data rate means a higher traffic volume.

4.4 Simulation Results

We report our simulation results in the order of evaluation variables listed above: (1) the effects of varying network sizes on accuracy and overhead of the clock-sync (TPSN) and event-sync (TSS) mechanisms, (2) the effects of varying node mobility levels, and (3) the effects of varying data rates. For each simulation scenario, we generate ten random cases, in which each case represents a different network topology and pairs of source-sink nodes. The synchronization error and overhead are obtained by running both the clock-sync (TPSN) and event-sync (TSS) on the same ten random cases per scenario. Each data

point in the simulation results shows the average values of these ten random cases in each scenario.

To verify that the analytical model (described in Chapter 3) is consistent with the simulation result, we check how well the actual synchronization error measured from the simulation matches with the expected synchronization error derived from our analytical models. If they match well, simulation results validate our analytical models. In order to compute the expected synchronization error from the analytical models, it requires plugging in the correct values for parameters (u, l, P, T, and d) in Equations 3.6 and 3.13. The correct values for these parameters: u (the average hop delay estimation error) and r (the average clock skew rate) can be obtained directly from the network topology and traffic volume settings in our simulation scenarios. The correct values for the parameters: l (the average node level in TPSN or the average path length in TSS), d (the average elapsed time for pair-wise synchronization), T (the synchronization period in TPSN), and P (the average packet inter-arrival time in TSS) can be observed during simulation. For example, to compute TPSN's E_{ext}, the values for l and d are collected during the simulation, and then multiplied with r (set to be 10^{-6}) to compute E_{ext}. Then, we can plug-in these parametric values into the analytical models to compute three individual error components E_{sync}, E_{int}, and E_{ext}. Furthermore, we can sum these three error components to compute *expected overall errors* $(E^{tpsn})'$ and $(E^{tss})'$.

Figure 4.1 & 4.2, 4.4 & 4.5 and 4.7 & 4.8 shows the synchronization error decomposed into three individual error components for the TPSN and TSS under different network sizes, node mobility levels, and data rates. Each plot contains the following five lines:

- Line (1) shows the actual simulation results of E^{tpsn} and E^{tss};

- Lines (2)-(4) show the expected E_{sync}, E_{int}, and E_{ext} obtained by applying the simulation's values to Equations 3.3 $\sim$ 3.5 and 3.10 $\sim$ 3.12 in the analytical models;

and

- Line (5) shows the expected synchronization errors $(E^{tpsn})'$ and $(E^{tss})'$ as the sums of the above three error components. Note that they are expected values different from the actual values (E^{tpsn} and E^{tss}) measured in the simulation.

By observing the similar trends and magnitudes between lines (1) and (5) in Figure 4.1 & 4.2, 4.4 & 4.5 and 4.7 & 4.8, we can check if the analytical models are consistent with the simulation results for both TPSN and TSS. Note that there will be small discrepancies between these two lines because $(E^{tpsn})'$ and $(E^{tss})'$ are the expected synchronization error computed from analytical model, whereas E^{tpsn} and E^{tss} are the actual synchronization error from simulation.

4.4.1 Impact of Network Size on Accuracy and Overhead

Figure 4.1 shows the synchronization error of TPSN under different network sizes. The actual synchronization error from the simulation is shown as line (1), and it exhibits a growing trend as the network size increases. This is expected given that a larger network size implies a higher level of the clock synchronization hierarchy, therefore, lengthens the path for a sync message to travel from the root node to a target node. The expected synchronization error derived from the analytical model is shown as line (5), and it also exhibits the same growing trend as the actual synchronization error from the simulation. Close similarity between lines (1) and (5) shows that simulation results are consistent with our TPSN analytical model. To gain better understanding on how network size affects TPSN's synchronization error, we analyze the changes in these three error components (E_{sync}, E_{int}, and E_{ext}) under different network sizes. E_{sync} is shown as line (2) in Figure 4.1. It is a zero line for reasons given at Section 4.1: i.e., the simulator is configured such that the forward link delay and the reverse link delay are perfectly symmetric, making

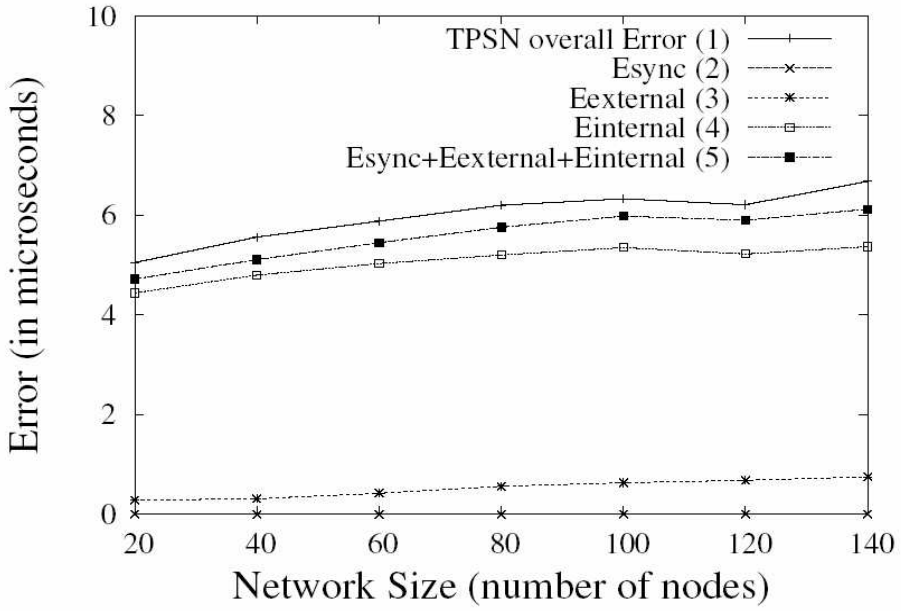

Figure 4.1: [Simulation] Synchronization errors for TPSN under different network sizes.

($u = 0$). E_{ext} is shown as line (3) in Figure 4.1. It also exhibits a growing trend with increasing network size. This is in accordance with Equation 3.4 – since a larger network size increases the depth of TPSN tree hierarchy (l), this also leads to higher E_{ext}. E_{int} is shown as line (4) in Figure 4.1. It also exhibits a growing trend with increasing network size. This is somehow unexpected by our analytical models. Given that the clock skew rate (r) and synchronization period (T) are fixed in the simulation setup, Equation 3.5 says that E_{int} should not change with increasing network size. After analyzing the simulation results, we have found an interesting relation between the network size and the synchronization period (T). This relation provides an explanation for the growing E_{int}. This relation can be explained as follows: (1) a larger network increases the average synchronization path length between the root node and other nodes down the hierarchy, (2) a longer path length leads to a higher probability of lost sync messages on that path, and (3) missing sync messages has the same effect as increasing the synchronization period (T).

Given the presence of this relation, we can derive the adjusted synchronization period (T') as a function of the message lost probability p (which is a function of the hierarchy level l) and original synchronization period (T):

$$T' \propto \frac{1}{1-p(l)} \cdot T \tag{4.1}$$

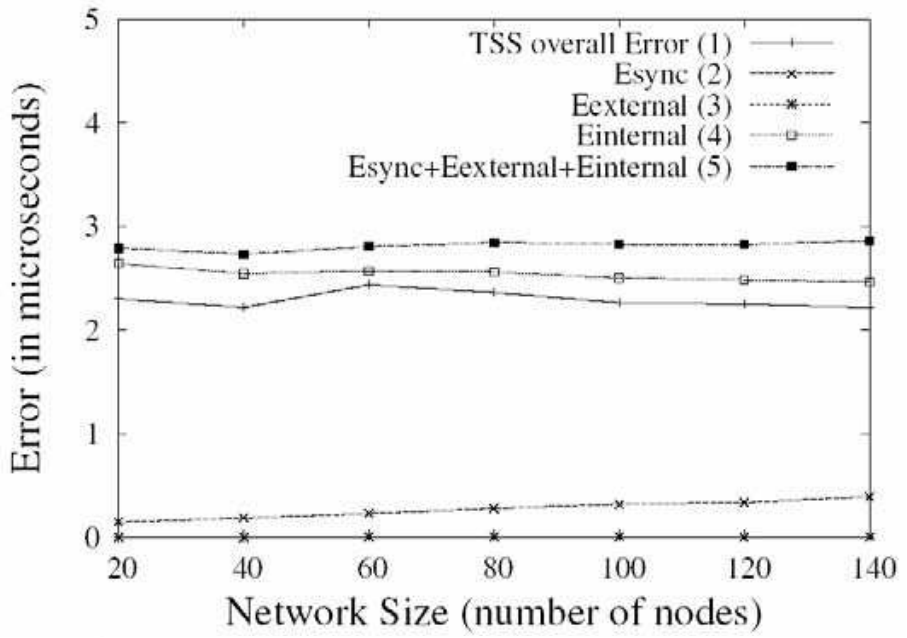

Figure 4.2: [Simulation] Synchronization errors for TSS under different network sizes.

Figure 4.2 shows the synchronization errors of TSS under different network sizes. The actual synchronization error from the simulation, shown as line (1), matches well with the model's expected synchronized error, shown as line (5). In other words, simulation results are consistent with our TSS analytical model. Next, we analyze the changes in three error components (E_{sync}, E_{int}, and E_{ext}) under different network sizes. E_{sync} and E_{ext} shown as lines (2) and (3) exhibit a growing trend, while E_{int} shown as line (4) is flat. This is in accordance with Equations 3.10 and 3.11, which say that E_{sync} and E_{ext} are affected by the length of data path from the source node to the sink node (l), which grows with

increasing network sizes. On the other hand, E_{int} is not affected by the changing network sizes, because the network size has no effect on any factors in Equation 3.12. Although both E_{sync} and E_{ext} grow with increasing network size, their scales ($< 0.1\mu s$) are one order of magnitude smaller than E_{int} ($2 \sim 3\mu s$). As a result, the overall TSS error is dominated by the flat E_{int}.

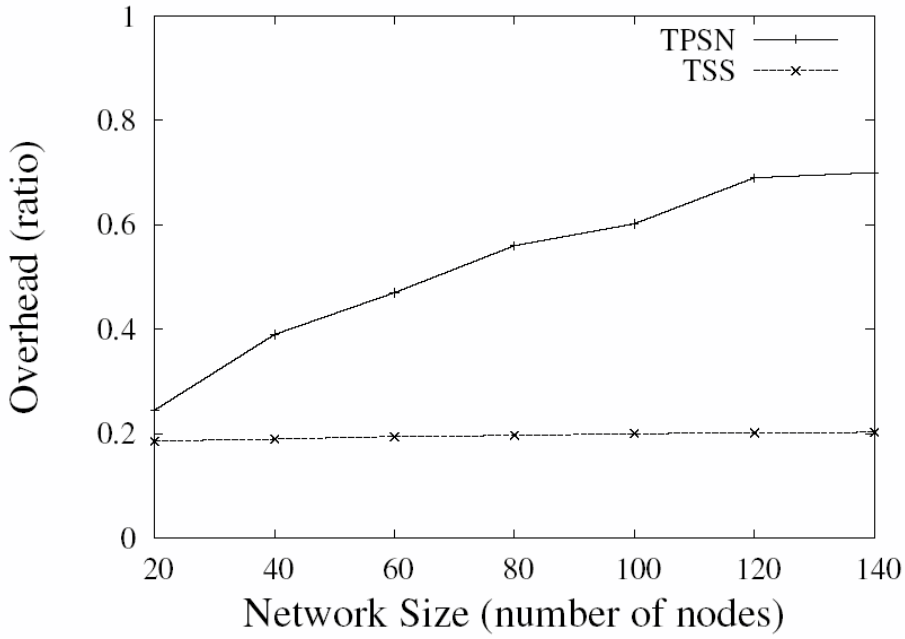

Figure 4.3: [Simulation] Protocol overhead of TPSN and TSS under different network sizes.

Figure 4.3 shows the protocol overhead ratios of TSS and TPSN under different network sizes. The *overhead ratio* is defined as the ratio between the amount of protocol overhead and the amount of data payload transfer. The overhead ratio of TSS increases only slightly with increasing network size, because TSS needs additional state establishment packets with longer data path. On the other hand, the overhead ratio of TPSN increases more dramatically than that of TSS. As the network size increases, the amount of synchronization packets in TPSN increases more rapidly than the amount of data packets.

Simulation results and analytical models have shown that TSS has smaller synchronization error, lower protocol overhead, and better scalability than TPSN under increasing network size.

4.4.2 Impact of Node Mobility Level on Accuracy and Overhead

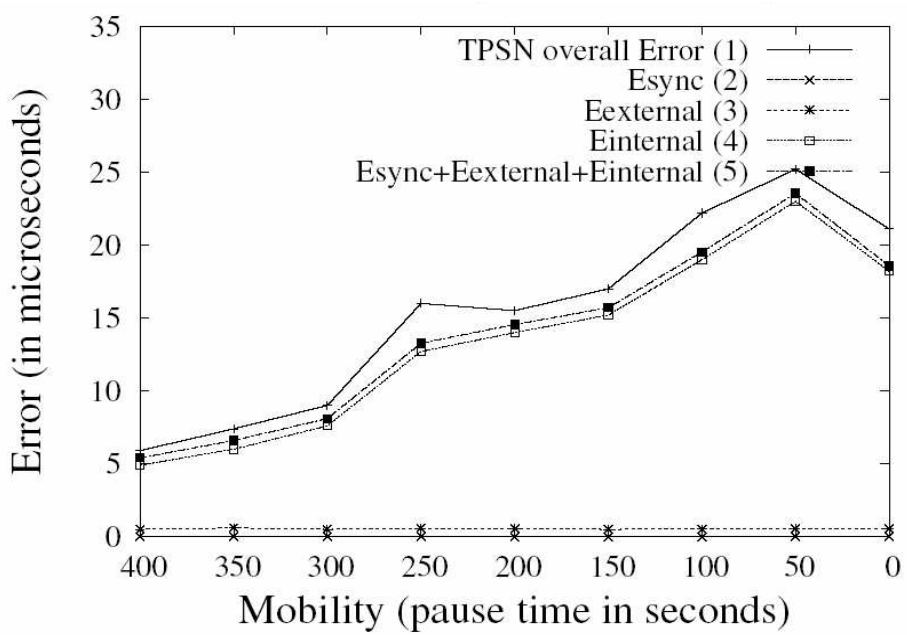

Figure 4.4: [Simulation] Synchronization errors for TPSN under different node mobility levels.

Figure 4.4 shows the synchronization error of TPSN under different node mobility levels. Node mobility levels are adjusted by changing the amount of pause time (from $0 \sim 400$ seconds) in the simulation. The actual synchronization error from simulation, shown as line (1), exhibits a growing trend as node mobility level increases. This is expected given that higher node mobility brings about more rapid change in the TPSN synchronization hierarchy. Before a new hierarchy is re-discovered by the mobile nodes that have moved away from their old neighbor nodes, they may not receive any synchro-

nization messages. As a result, their local clocks may continue to drift out-of-sync. Next, we apply the analytical models to help explain the effects of different node mobility levels on TPSN's synchronization error. We decompose the overall synchronization error into three error components (E_{sync}, E_{int}, and E_{ext}) shown in Figure 4.4. For the reason given at the end of Section 4.1, E_{sync} , shown as line (1), is a zero line. E_{ext} shown as line (3) exhibits a relatively flat line. This is expected given that a higher level of node mobility does not change the size of the synchronization hierarchy l. E_{int} shown as line (4) exhibits a growing trend with increasing levels of node mobility. This is also expected given the following reasoning: higher node mobility implies higher probability of broken hierarchy, which in turn leads to higher probability that nodes may fail to be synchronized with upper level nodes. This has the same effect as raising the average synchronization time interval T, causing higher synchronization error. An interesting phenomenon in TPSN is that when the node mobility level moves to the high end of the extreme (pause times = $50 \sim 0$ seconds), the synchronization error actually *falls*. This phenomenon can be explained as follows: since high mobility enables the root node to encounter more one-hop neighbor nodes over the course of its movements (high mobility in non-root nodes also increases the probability that they will encounter the root node); therefore, mobility helps to spread the root node's global clock to a larger number of level-1 neighbor nodes. This is an example where mobility can sometimes be a positive factor.

Figure 4.5 shows the synchronization error of TSS under different node mobility levels. The actual synchronization error from simulation is shown as line (1). It exhibits a two-step behavior - a flat line under low node mobility, but changes to a steep line under high node mobility. This two-step behavior can be explained by decomposing the overall synchronization into three error components (E_{sync}, E_{int}, and E_{ext}) and analyzing them. E_{sync} shown as line (2), exhibits a decreasing trend with increasing node mobility levels. The reason is the *restriction on the routing mechanism* - when the network is dynamic un-

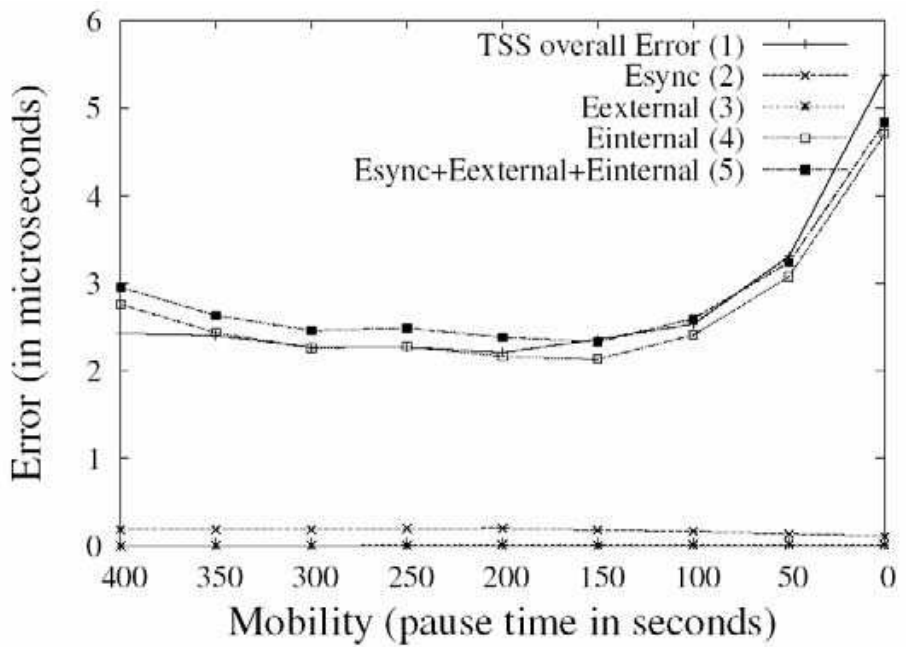

Figure 4.5: [Simulation] Synchronization errors for TSS under different node mobility levels.

der high node mobility, data is more likely to be dropped before reaching the sink node. Therefore, the average data path length (counting dropped packets) will decrease. Simulation results confirm this observation. E_{ext} shown as line (3), exhibits a slight growing trend with increasing node mobility. This is in accordance with Equation 3.11, which says that the positive factor influencing E_{ext} is the hop delay (d). The increase of E_{ext} is mainly due to a larger d induced in a dynamic network. This is due to data packets spending more time waiting on intermediate nodes that are searching for the next hop to forward the data packets. Simulation results confirm this observation. In addition it shows that the effect of a larger d, which raises E_{ext}, dominates the effect of shorter l, which lowers E_{ext}. As a result, E_{ext} exhibits a growing trend. E_{int} also exhibits a growing trend with increasing node mobility. This growth is mainly due to a larger data packet inter-arrival time (P). The rise in packet inter-arrival time is caused by frequent switching of routing

paths. Switching of routing paths is a result of intermediate nodes repeatedly moving in and out of routing paths, making use of previously established time-sync states on nodes. Since E_{int} is the dominant factor, the overall sync error exhibits the growing trend.

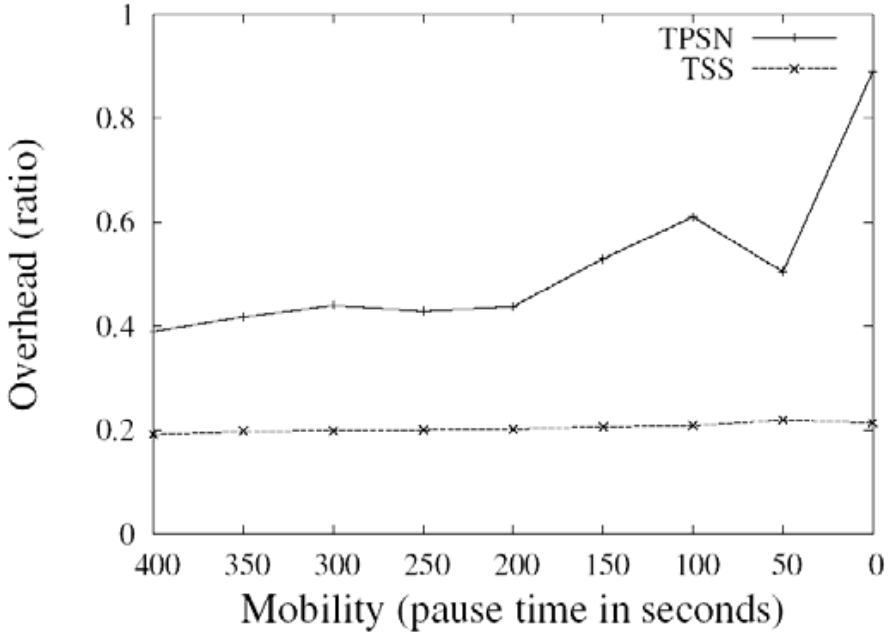

Figure 4.6: [Simulation] Protocol overhead of TPSN and TSS under different network sizes.

Figure 4.6 shows the protocol overhead ratios of TSS and TPSN under different levels of node mobility. Since higher node mobility results in higher frequency of topology changes, extra packets are sent to reconstruct the new TPSN hierarchy. As a result, the protocol overhead is raised. In contrast, TSS does not maintain any synchronization hierarchy. TSS has to re-establish time-sync states on the new routing paths when intermediate nodes move out of the routing paths. However, the amount of overhead increased in TSS is not as significant as TPSN.

Simulation results and analytical models have shown that TSS has smaller synchronization error and lower protocol overhead than TPSN under increasing network dynam-

ics. Although TSS has smaller synchronization error than TPSN, TSS's synchronization error grows steeper than TPSN under high node mobility, making TSS less scalable than TPSN.

4.4.3 Impact of Date Rate on Accuracy and Overhead

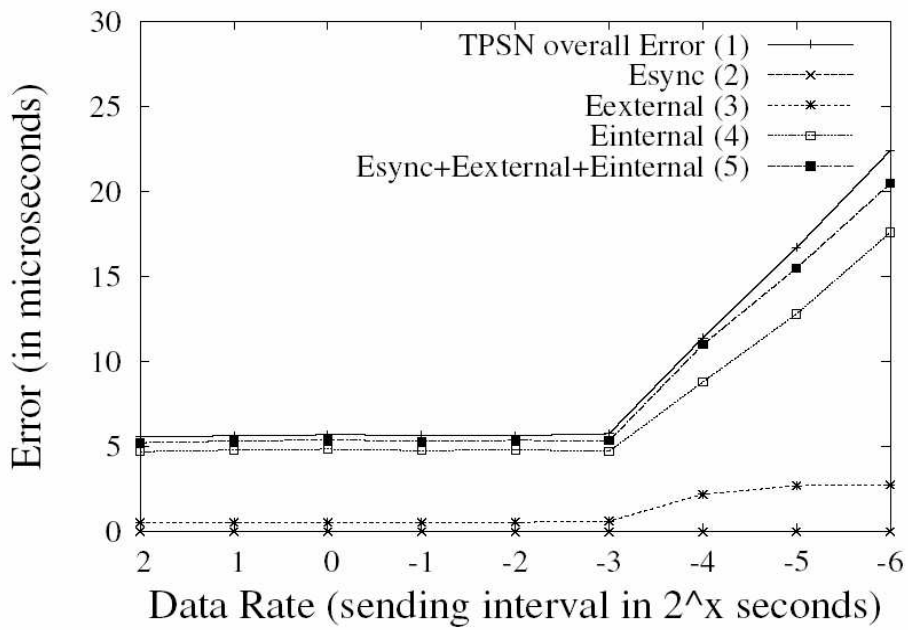

Figure 4.7: [Simulation] Synchronization errors for TPSN under different data rates.

Figure 4.7 shows the synchronization error of TPSN under different data rates. Data rates are adjusted by changing the data packet sending rates at source nodes. The actual synchronization error from the simulation, shown as line (1), exhibits a two-step behavior. These two steps can be explained as follows. High traffic volume eventually leads to network congestion and packet loss. In the simulation scenario, this *congestion point* is the point connecting these two steps (i.e., the packet sending interval is approximately 2^{-3} second). If the lost packets contain sync messages, nodes will miss synchronization rounds. Missing synchronization rounds has the same effect as increasing the sync period

(T). Equation 3.6 says that increasing T leads to higher synchronization error. Next, we decompose the overall synchronization error into three error components (E_{sync}, E_{int}, and E_{ext}) shown in Figure 4.7. Again E_{sync} is a zero line for the reason given at the end of Section 4.1. E_{ext} is shown as line (3), and it exhibits a growing trend when data rates go beyond the congestion point. This is reasonable because network congestion lengthens the media access time and the packet hop delay (d). E_{int} is shown as line (4), and it also exhibits the growing trend after the congestion point. The reason has already been given above - congestion induces dropped packets and lost synchronization messages, which can be translated into increase of the synchronization period (T).

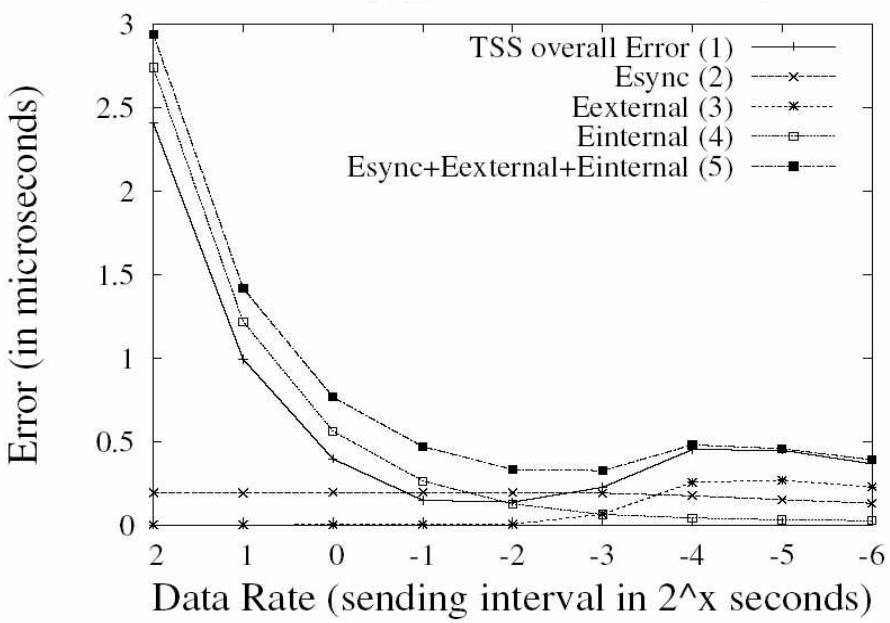

Figure 4.8: `[Simulation]` Synchronization errors for TSS under different data rates.

Figure 4.8 shows the synchronization error of TSS under different data rates. Interestingly, the actual synchronization error exhibits a decreasing trend before the congestion point, and remains flat after the congestion point. We can explain this two-step behavior from the analytical models and by decomposing the overall synchronization error into

three error components (E_{sync}, E_{int}, and E_{ext}). E_{sync} is shown as line (2), and it is flat because the data path length does not change with increasing data rates. E_{ext} is shown as line (3), and it also remains flat before the congestion point, but grows after the congestion point. We can explain this two-step behavior by looking at Equation 3.11 and in combination with the fact that network congestion increases the queue length and the hop delay (d). E_{int} is shown as line (3), and it exhibits a downward trend. This can be explained by looking at Equation 3.12 - a high data rate means a small packet sending interval, and this leads to a shorter packet inter-arrival time (P) at the sink node! When data rates are low with no congestion, E_{int} falls linearly because the amount of decrease in P is linear to the data sending interval. However, when data rates are high with congestion presence, P does not hold a linear relation with the data sending interval, because network congestion can slow down the actual data receiving rate. Therefore, E_{int} decreases at a slower rate. The overall synchronization error in TSS drops under low data rates because E_{int} dominates the overall error. This explains the downward trend for the first step in E^{tss}. However, under high data rates, E_{int} is small and E_{ext} becomes the dominant factor in E^{tss}. This explains the slight-upward trend in the second step of E^{tss}.

Figure 4.9 shows the protocol overhead ratios of TSS and TPSN under different data rates. Since TPSN exchanges synchronization messages at a fixed time interval independent of data traffic volume, the overhead ratio will fall as the data rate increases. However, it will stop dropping when data rates increases beyond the congestion point. In comparison, TSS exhibits a flat line because the protocol overhead is piggybacked on the data packet, fixing its protocol overhead ratio.

Although TSS has a smaller synchronization error than TPSN, TPSN has a significantly lower overhead under high data rates.

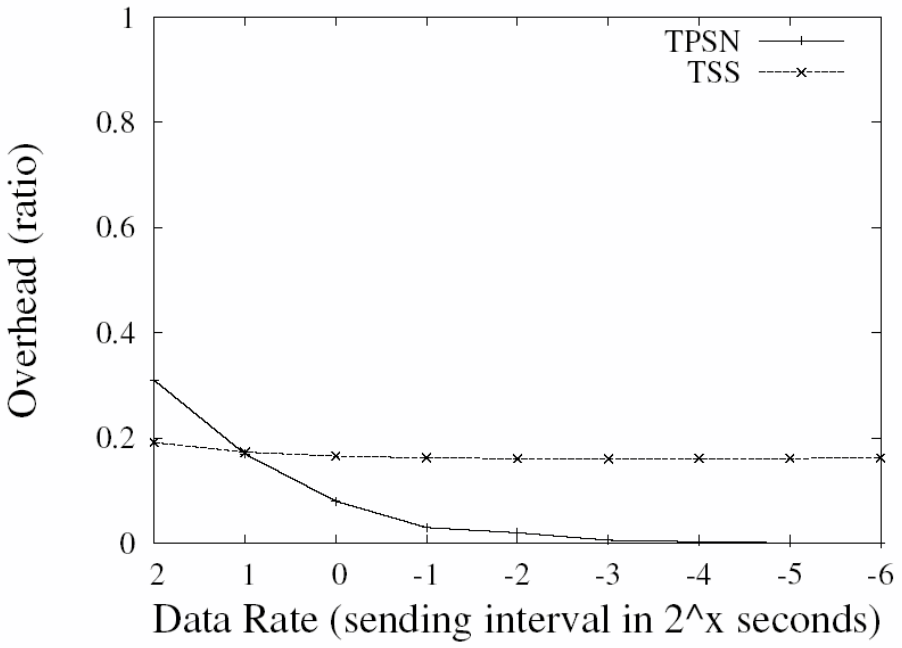

Figure 4.9: [Simulation] Protocol overhead of TPSN and TSS under different network sizes.

Chapter 5

Taroko Mote Experiments

Having shown that our analytical models are consistent with the *ns-2* simulation results, we conduct real experimental evaluation of TPSN and TSS protocols by implementing and running TPSN and TSS over a small testbed consisting of Taroko motes. Using real sensor hardware provides a more realistic comparison between TPSN and TSS. We describe the details of this experiment setup, including description of Taroko motes and evaluation variables (network size and traffic volume). We also compare the performance results between those from simulation and those from NTU Taroko sensor nodes.

5.1 Taroko Mote Platform

We conducted our experiments on Taroko motes. The Taroko mote is a Zigbee-based ultra-low power wireless sensor platform suitable for sensor network applications as shown in Figure 5.1. Taroko mote has an 8MHz microcontroller and 10KB on-chip RAM enabling more sophisticated applications running on it. With a Zigbee radio, each Taroko mote has a radio bit rate at 250 kbps, and radio range of up to 30 meters [1] for wireless communication. For a local clock, Taroko mote uses a 32768 Hz clock ($\sim 30\ \mu s$ preci-

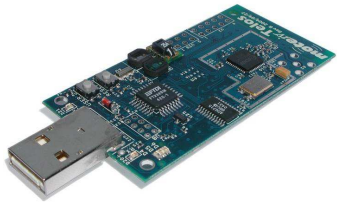

Figure 5.1: Taroko mote: a Zigbee-based ultra-low power wireless sensor module. (physical size $6.5 \times 3.5 \times 0.5cm$).

sion) with a clock skew rate of 20×10^{-6} to 100×10^{-6} [2]. Because of its small form factor, low-power design, and ease of development, Taroko mote is adequate for sensor network applications.

5.2 Taroko Mote Experimental Setup

The experiment duration is 300 seconds. The data used are restricted to those collected after 50 seconds. Similar to Section 4.1, this avoids taking the start-up time instability affecting the experimental results. For evaluation parameters, the base case is defined to have 4 nodes in a linear topology as shown in Figure 5.2, and node 1 and node 4 are sink and source respectively. Each node sends a 100 bytes packet every 5 seconds. Unless specified otherwise, these are the default values for the parameters.

5.3 Evaluation Metrics

Two identical metrics in simulation are used in the experiments to evaluate the performance of clock synchronization and event synchronization: *synchronization error* and *overhead*. Synchronization error represents the difference between actual data generation time and estimated data generation time, whereas overhead represents the traffic produced by the synchronization protocols in proportion to the total data traffic.

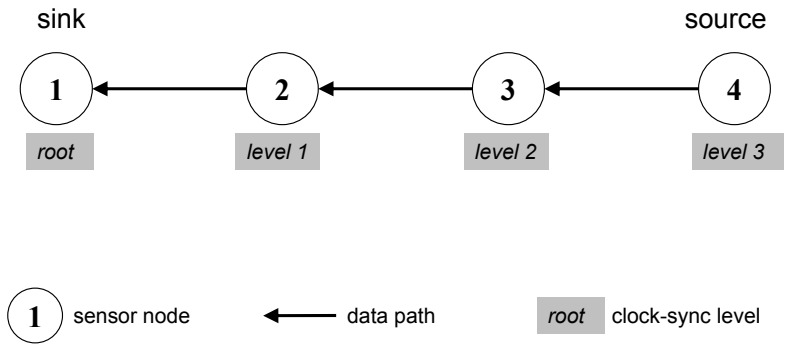

Figure 5.2: Linear topology of 4 nodes. Node 1 and node 4 are sink and source respectively.

5.4 Evaluation Variables

To compare the error and overhead of these two time synchronization mechanisms, different scenarios ran on Taroko motes with varying *network sizes* and *data rates*.

- *Network Sizes*: To vary the network sizes, the number of nodes was changed from 2 to 7 (1 to 6 hops) with incremental steps of 1 node. As shown in Figure 5.2, node 1 is always the sink node and the source node is the node with largest ID.

- *Data Rates*: To vary the data rates, we adjusted the packet sending rates at the source nodes, from a fixed-size packet every 4 (2^2) seconds to every 0.0625 (2^{-4}) second. A higher data rate means a higher traffic volume.

5.5 Taroko Mote Experimental Results

We report our experimental results in the order of evaluation variables listed above: (1) the effects of varying network sizes on accuracy and overhead of the clock-sync (TPSN) and event-sync (TSS) mechanisms, and (2) the effects of varying data rates.

5.5.1 Impact of Network Size on Accuracy and Overhead

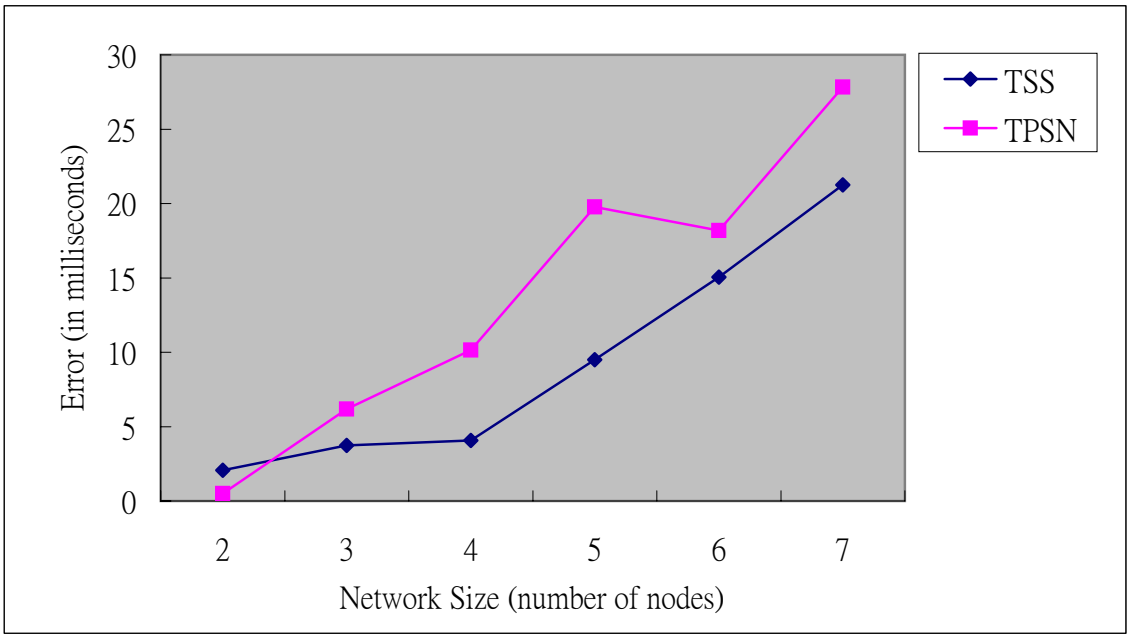

Figure 5.3: [Taroko Mote Experiment] Synchronization errors for TPSN and TSS under different network sizes.

Figure 5.3 shows the synchronization error of TPSN and TSS under different network sizes for the Taroko motes. The synchronization error of TPSN and TSS are both at the millisecond level. The large magnitude of error is caused by Taroko mote's *coarse clock precision*. In comparison with Mica2's 0.25 μs minimum crystal granularity [10], Taroko mote's 30 μs clock precision is much coarser. Therefore, collecting timestamps at various times during the synchronization protocols involves higher clock round-up or round-down error, magnifying the scale of synchronization error. Besides, TSS shows smaller synchronization error than TPSN as network size increases, in accordance with the observation mentioned in Section 4.4.

Figure 5.4 shows the protocol overhead ratios of TPSN and TSS under different network sizes. The overhead ratio of TPSN grows at a faster rate as the network size increases

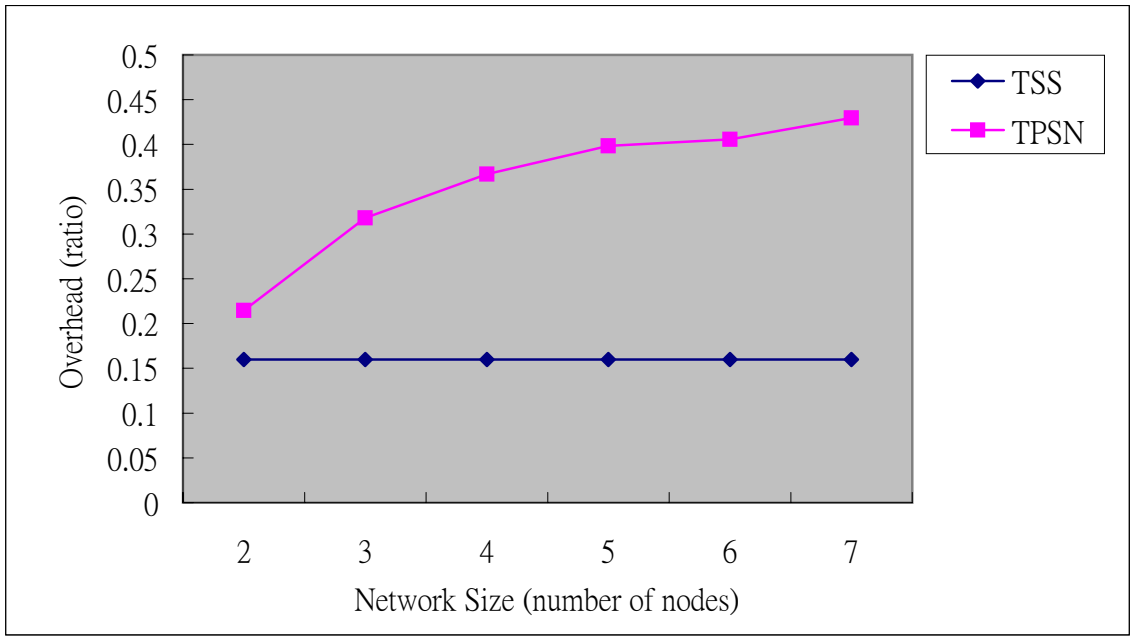

Figure 5.4: [Taroko Mote Experiment] Protocol overhead for TPSN and TSS under different network sizes.

because of rapid increase of synchronization packets as the network size increases. For TSS, it shows a flat line due to the constant size of synchronization information contained in each data packet.

Taroko mote experimental results have shown that TSS has a smaller synchronization error and a lower protocol overhead than TPSN under increasing network size. This matched our observation in simulation results.

5.5.2 Impact of Data Rate on Accuracy and Overhead

Figure 5.5 shows the synchronization error of TPSN and TSS under different data rates. On one hand, the synchronization error of TPSN exhibits similar behavior to the simulation results. The line remains level until the congestion point (at the packet sending interval approximately 2^{-3} second), and from that a sudden raise seems to grow dramati-

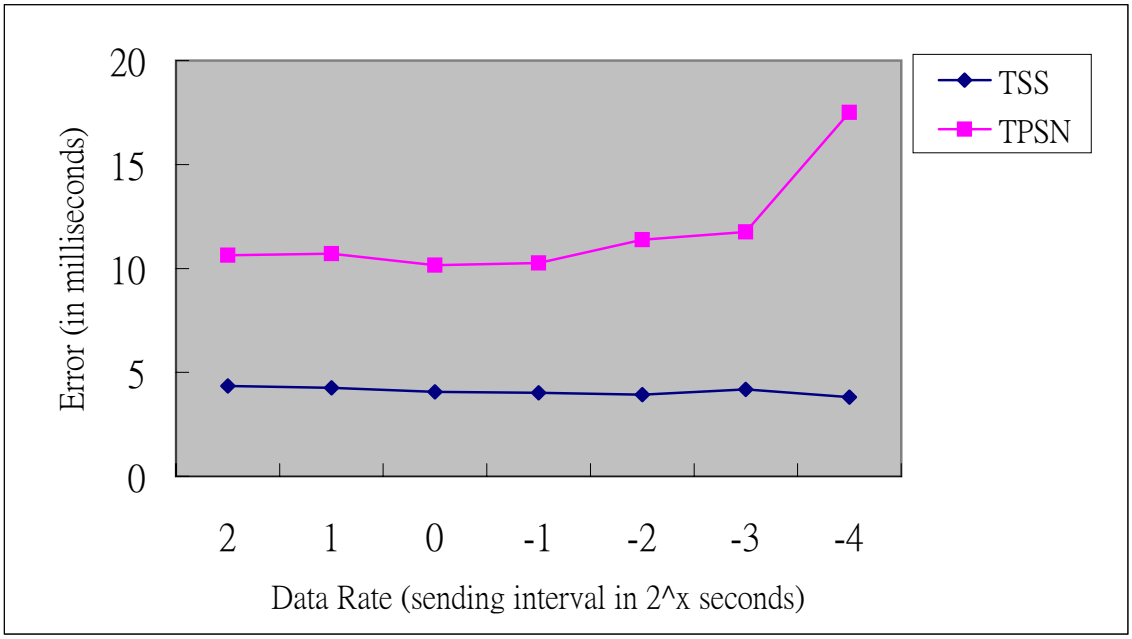

Figure 5.5: [Taroko Mote Experiment] Synchronization errors for TPSN and TSS under different data rates.

cally. On the other hand, the synchronization error of TSS presents almost flat. Again, due to same cause noted in Subsection 5.5.1, the pair-wise synchronization error E_{sync} dominates the overall error T^{tss}, and alleviates the impact of data rate on the synchronization error.

Figure 5.6 shows the protocol overhead ratios of TPSN and TSS under different data rates. Similar to the simulation results shown in Figure 4.3, because the protocol overhead is piggybacked on the data packet, the overhead ratio of TSS displays a fixed overhead ratio. And the overhead ratio of TPSN drops with increasing data rate since TPSN exchanges synchronization messages independent of data traffic volume.

Taroko mote experimental results have shown that the synchronization error of TSS is smaller than of TPSN, yet the overhead ratio of TPSN is lower under high data rates.

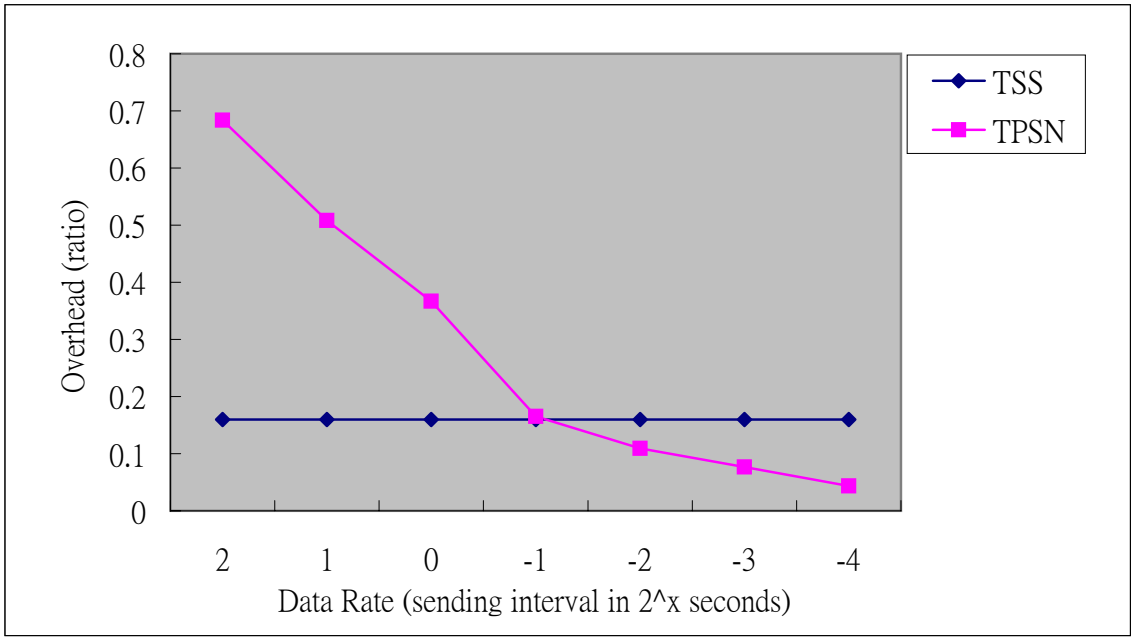

Figure 5.6: `[Taroko Mote Experiment]` Protocol overhead for TPSN and TSS under different data rates.

Chapter 6

Conclusions and Selection Guideline

In conclusion, we achieve in (1) modeling the average error of two time synchronization mechanisms, TPSN and TSS, (2) validating the analytical models with thorough simulations and real sensor experiments, and (3) deriving the following selection guideline for choosing a suitable time synchronization protocol based on the network size, node mobility level, and data rate.

- When network size is large, TSS is dominantly better both in terms of accuracy and overhead.
- When mobility level is high and energy consumption is critical, TSS will be the choice. If energy consumption is not critical, the choice should depend on the actual mobility level. TSS appears to be better in the error to reference clock. However, the error of TSS has a steep growing trend. When the mobility level goes beyond the parameters used for the experiments, the error of TSS could potentially be worse than that of TPSN.
- When data rate is high and energy consumption is critical, TPSN will be the choice. If energy consumption is not critical, the error of TSS is lower and scales better.

TSS will be the choice in this case.

Although TSS outperforms TPSN in accuracy under most of the cases, TSS has a fundamental limitation. 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, the generation times of events that go to different sink nodes are not synchronized. If the application requires events to be labeled using a global reference clock, TSS is not applicable.

Appendix A

Publication of Jr-ben Tian

Below is a list of publications that I have achieved in the study of master program:

1. Keng-hao Chang, Shih-yen Liu, Jr-ben Tian, Hao-hua Chu, Cheryl Chen, **"Dietary-Aware Dining Table - Tracking What and How Much You Eat"**, in *Proceedings of Workshop on Smart Object Systems*, in conjunction with the Seventh International Conference on Ubiquitous Computing (ACM UbiComp 2005), Tokyo, Japan, September 11, 2005, pages 61-68

Bibliography

[1] http://www.ubec.com.tw/services/faqs.html.

[2] http://www.microcrystal.com/.

[3] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E.Cayirci. Wireless sensor networks: A survey. *Computer Networks*, 38:393–422, 2002.

[4] Archana Bharathidasan and Vijay Anand Sai Ponduru. Sensor networks: an overview. Technical report, UC Davis.

[5] Lee Breslau, Deborah Estrin, Kevin Fall, Sally Floyd, Ahmed Helmy, Polly Huang, Steven McCanne, Kannan Varadhan, Ya Xu, Haobo Yu, and VINT Project. Advances in network simulation. *IEEE Computer*, 33(5):59–67, May 2000.

[6] Alberto Cerpa, Jeremy Elson, Michael Hamilton, and Jerry Zhao. Habitat monitoring: Application driver for wireless communications technology. In *Proceedings of International Conference ACM SIGCOMM*, April 2001.

[7] Jeremy Elson, Lewis Girod, and Deborah Estrin. Fine-grained network synchronization using reference broadcasts. In *Proceedings of the 5th symposium on Operating System Design and Implementation (OSDI))*, December 2002.

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

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

[10] Saurabh Ganeriwal, Ram Kumar, and Mani B. Srivastava. Timing-sync protocol for sensor networks. In *Proceedings of 1st ACM Conference on Embedded Neworked Sensor Systems (SenSys)*, November 2003.

[11] Chalermek Intanagonwiwat, Ramesh Govindan, and Deborah Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking (MobiCOM)*, August 2000.

[12] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of ACM*, 21(7):558–565, 1978.

[13] Miklos Marotia, Branislab Kusy, Gyula Simon, and Akos Ledeczi. The flooding time synchronization protocol. Technical Report ISIS-04-501, Institute for Software Integrated Systems, Vanderbilt University, 2004.

[14] Kay Römer. Time synchronization in ad hoc networks. In *Proceedings of ACM symposium on Mobile Ad-Hoc Networking and Computing (MobiHoc)*, October 2001.

[15] Zhuohui Zhang. Investigation of wireless sensor networks for precision agriculture. Paper Number 041154, ASAE Annual Meeting, 2004.