Error compensation algorithm in wireless sensor networks synchronisation

Fuqiang Wang

Key Laboratory of Industrial Informatics,
Shenyang Institute of Automation,
Chinese Academy of Sciences,
Shenyang, China
and
Shandong Computer Science Center,
Jinan, China
Email: wangfq@keylab.net

Peng Zeng and Haibin Yu*

Key Laboratory of Industrial Informatics,
Shenyang Institute of Automation,
Chinese Academy of Sciences,
Shenyang, China
Email: zp@sia.cn
Email: yhb@sia.cn
*Corresponding author

Yang Xiao

Department of Computer Science,
The University of Alabama,
101 Houser Hall, P.O. Box 870290
Tuscaloosa, AL 35487 0290, USA
Email: yangxiao@ieee.org

Abstract: Time synchronisation has been extensively used in wireless sensor networks in many contexts, such as data fusion, TDMA schedules and synchronised sleep periods. Precision is the basic requirement for time synchronisation in wireless sensor networks. Existing time synchronisation methods were not designed to deal with dynamic environment changes. In this paper, we propose an error compensation algorithm which is designed to compensate errors caused by hardware clock drift and environment changes. The algorithm works according to a level hierarchy: the nodes in each level have the same hop and linear regression is adopted to analyse clock drift. The compensation mechanism decreases the instability caused by crystal oscillators and eliminates errors accumulated during running time. The algorithm is implemented on the SIA2420 platform using TinyOS and the results show the reliability of our algorithm.

Keywords: WSNs; wireless sensor networks; time synchronisation; clock drift; compensation algorithm.


Biographical notes: Fuqing Wang holds a PhD degree in Mechatronic Engineering from Shenyang Institute of Automation (SIA), Chinese Academy of Sciences. He is a Research Associate at Shandong Computer Science Center. His research interests are in the areas of time synchronisation in wireless sensor networks and hopping technology.

Peng Zeng holds a PhD degree in Mechatronic Engineering from Shenyang Institute of Automation (SIA), Chinese Academy of Sciences. He is a Professor at Shenyang Institute of Automation, Chinese Academy of Sciences. His research has involved industrial communication and wireless sensor networks.
1 Introduction

Recently, Wireless Sensor Networks (WSNs) have drawn great research attention. Thousands or even millions of wirelessly connected nodes, which are capable of computation, communicating and sensing, are deployed in target environments, interacting with the environment and communicating with each other frequently. As one of the key technologies in WSNs, time synchronisation plays an important role in node localisation, low power listening, data fusion, TDMA, synchronised hopping systems, etc.

Although each sensor node is equipped with a hardware clock, these hardware clocks cannot usually be used directly as they suffer from severe drift. No matter how well these hardware clocks are calibrated at deployment, they will ultimately exhibit a large skew. To get an accurate common time, nodes need to exchange messages from time to time in order to adjust their clock values.

Many existing synchronisation protocols share a basic design: a server periodically sends a message containing its current clock value to a client. Since most networks are very closely associated with their applications, the protocols intended for synchronisation are different from each other in some aspects; three kinds of synchronisation modes are used to classify existing protocols: receiver–receiver, sender–receiver and one-way time protocols. Reference Broadcast Synchronisation (RBS) works in the receiver–receiver mode and any node can communicate directly with other nodes in the network. Such an approach removes the risk of master node failure. Therefore, these classes of protocols are more flexible but also more uncontrollable. Many protocols, such as Timing-Sync Protocol for Sensor Network (TPSN; Ganeriwal et al., 2003), Tiny-Sync and Mini-Sync (Sichitiu and Veeraritiplan, 2003), TSynch (Dai and Han, 2004), LTS (Greunen and Rabaey, 2003), Global Clock Synchronisation (AD; Qun and Daniela, 2004) and Time-Stamp Synchronisation (TSS; Kay, 2001), employ a two-way (sender–receiver) pattern to maintain synchronisation. By measuring the total roundtrip-time of the two packets, the client can estimate the one-way latency. This allows for more accurate synchronisation by accounting for the time that elapses between the server’s creation of a time-stamp and the client’s reception of it, but it results in more messages being exchanged which correspond to greater energy consumption. One-way time synchronisation has been the most popular method used in recent years. A simple one-way message suffices if the typical latency from server to client is small compared to the desired accuracy as in Delay Measurement Time Synchronisation (DMTS; Ping, 2003). However, the linear regression is adopted to analyse the clock drift between nodes and the network can become more stable, and more accurate time synchronisation and the Flooding Time Synchronisation Protocol (FTSP; Maroti et al., 2004) works well with this mechanism in wireless sensor networks.

Although great progress has been made in this area, there is still much space for improvement. In this paper, we propose an error-compensation algorithm which is designed to compensate for the error caused by hardware clock drift or environmental changes. The algorithm works according to a level hierarchy: the nodes in each level have the same hop and linear regression is adopted here to analyse clock drift. The compensation mechanism decreases the instability caused by crystal oscillators and eliminates errors accumulated during running time. The algorithm is implemented on the SIA2420 platform using TinyOS and the results show the reliability of our algorithm.

The remainder of this paper is organised as follows. In Section 2, the related work on synchronisation in sensor networks is exhibited. The error-compensation algorithm of time-synchronisation used in this paper is presented in Section 3. In Section 4, the testbed of our implementation is introduced. In Section 5, the experimental results are given, and the conclusion is presented in Section 6.

2 Related work

Clock synchronisation had been studied long before the advent of WSNs. The classic solution is an atomic clock, such as in the Global Positioning System (GPS; Zhu, 1994). However, GPS-based clock acquisition schemes exhibit some weaknesses: GPS is not ubiquitously available (for example, underwater, indoors and under foliage) and requires a relatively high-power receiver, which is not possible with small, cheap sensor nodes. These shortcomings motivated the development of software-based approaches to achieve in-network time synchronisation.
Classical clock synchronisation algorithms rely on the ability to exchange messages at a high rate, which may not be possible in WSNs. Traditional time synchronisation algorithms like the Network Time Protocol (NTP) are not suitable for a wireless sensor environment (Guanlin et al., 2005), since wireless sensor networks pose numerous challenges of their own, including limited energy and bandwidth, limited hardware, latency and unstable network conditions caused by the mobility of sensors, dynamic topology and multi-hopping.

Sensor networks require sophisticated algorithms for clock synchronisation because the hardware clocks in sensor nodes are often simple and may experience significant drift. Also, in contrast to wired networks, the multi-hop character of wireless sensor networks complicates the problem, as one cannot simply employ a standard client/server clock synchronisation algorithm.

As research in sensor networks has evolved in recent years, many different approaches for time synchronisation have been proposed. Römer presents a system in which events are time-stamped with the local clock (Kay, 2001). When such a time-stamp is passed to another node, it is converted to the local time-stamp of the receiving node. The aim here is not to synchronise the sensor node clocks but to generate a correct chronology of events. A scheme for sensor networks based on this model was proposed by Kay (2001). The algorithm proposed by Kay (2001) is only initiated when events take place in the network. Therefore, such a scheme cannot be extended to scenarios requiring a sensor node clock. For example, a scenario in which the actual time of occurrence of an event is important or in which a sensor node clock is used to successfully run MAC protocols.

A more complex model involves the maintaining of relative clocks. In this model, though every node maintains an individual clock, these clocks are not synchronised. Instead, every node stores information about the relative drift between its clock and the clock of every other node in the network. A scheme based on this model is RBS (Elson et al., 2002). In RBS, sensor nodes periodically send beacon messages to their neighbours using the network’s physical layer broadcast. Recipients use the message’s arrival time as a point of reference for comparing their clocks. The offset between any pair of nodes receiving the beacon is calculated by exchanging their local time-stamps. This scheme successfully removes all possible sources of error except for the variability in processing delay at the receiver.

Another model requires every node to maintain a clock that is synchronised with respect to a reference node in the network. The aim here is to maintain a global and a unique timescale throughout the network. A scheme based on this model is TPSN (Ganeriwal et al., 2003). The TPSN algorithm elects a root node and builds a spanning tree of the network during the initial level discovery phase. In the synchronisation phase of the algorithm, nodes synchronise with their parents in the tree by a two-way message exchange. Using the time-stamps embedded in the synchronisation messages, the child node is able to calculate the transmission delay and the relative clock offset. However, TPSN does not compensate for clock drift, which makes frequent resynchronisation mandatory. In addition, TPSN incurs a high communication overhead since a two-way message exchange is required for each child node.

The FTSP (Maroti et al., 2004) synchronises the network by successively broadcasting the synchronisation messages using MAC layer time-stamping and performing skew compensation based on a linear regression. Each node uses a linear regression table to convert back and forth between the local hardware clock and the clock of the reference node. The root node is dynamically elected by the network based on the smallest node identifier. After initialisation, a node waits for a few rounds and listens for synchronisation beacons from other nodes. Each node that is sufficiently synchronised to the root node starts broadcasting its estimation of the global clock. If a node does not receive synchronisation messages during a certain period, it will declare itself the new root node. FTSP achieves a higher level of synchronisation accuracy than either RBS or TPSN.

The paper by Sommer and Wattenhofer (2008) presents a clock synchronisation algorithm with drift compensation to implement the symmetric clock synchronisation in sensor networks. By using a simple moving average filter, a symmetric synchronisation error can be achieved. A recursive formula is used to address the uncertainty in the measurement of the change of the offset and a prototype implementation of the algorithm shows the error between estimation and the real reference clock value is very small (1–2 seconds).

The Gradient Time Synchronisation Protocol (GTSP; Sommer and Wattenhofer, 2009) provides clocks which are accurately synchronised between neighbours. GTSP works in a completely decentralised fashion, with every node periodically broadcasting its time information. A logical clock is calibrated using synchronisation messages received from its direct neighbours. The algorithm requires neither a tree topology nor a reference node, which strengthens it against link and node failures.

Furthermore, there are many related papers in this area (e.g. Al-Karaki et al., 2006; Araz and Qi, 2006; Deng et al., 2006; Huang et al., 2006; Wang et al., 2006; Xiao et al., 2006; Zheng et al., 2006; Du et al., 2007; Hu et al., 2007; Liu et al., 2007; Du et al., 2008; Hu et al., 2008; Liu et al., 2009; Galloway et al., 2010; Liu et al., 2010).

The protocols mentioned above have provided many choices of wireless sensor networks but great efforts must be made to apply this to many complex environments such as industrial workshops and fields where temperatures change frequently. Through our advancement of the one-way time synchronisation mechanism, the instant changes of the hardware clocks were eliminated and the occasional errors between nodes were detected and compensated for in order to attain more stable synchronisation.
3 Error compensation algorithm

In this section, we describe our compensation algorithm for the time synchronisation protocol. The basic idea is to use linear regression to estimate the drift of the receiver clock with respect to the sender clock and to use error compensation to eliminate errors of clock offset caused by message transmission and regression deviation.

In a network consisting of sensor nodes with perfect hardware clocks (no drift), time progresses at the same rate throughout the network and, once the offset between child nodes and the root node is calculated, the nodes agree on a common global time. However, real hardware clocks exhibit relative drift in the order of up to 100 ppm, which leads to continually increasing synchronisation errors between the nodes.

Therefore, a high frequency synchronisation process is adopted to guarantee certain bounds for the synchronisation error. However, exchanging synchronised messages more frequently causes increased energy consumption, which is unsuitable for energy-constrained sensor networks. Clock drifts between nodes should be compensated for in order to achieve more stable synchronisation with fewer messages exchanged. In our clock synchronisation algorithm we propose an error compensation algorithm to predict and compensate clock drift caused by either the node itself or the environment. Therefore, the precision of the demands corresponds to less energy consumption.

3.1 Logic clock

Since other hardware components may depend on a continuously running hardware clock, its value should not be adjusted manually. Instead, a logical clock is maintained as a function of the local hardware clock while the system is running.

A linear model is used here to set up the logical clock model. We assume that the root for the network synchronisation has been established and we denote its hardware time as \( T_{global} \) which acts as the global time of the network. Hardware clocks in all nodes except the root node are depicted as \( T_{Local}(i) \) and the logical clock of each node as \( T_{Logical}(i) \). If all nodes in a network have perfect synchronisation, the logical time of each node will be consistent with the global time of the root node.

We can calculate the linear model of the global time and the local time as follows:

\[
T_{local}(t) = \beta_{local} + \alpha_{local} t
\]

\[
T_{global}(t) = \beta_{global} + \alpha_{global} t
\]

Therefore, the relation between the global time and the local time is:

\[
T_{global}(t) = \eta * T_{local}(t) + \Delta T
\]

Here:

\[
\eta = \frac{\alpha_{global}}{\alpha_{local}}, \quad \Delta T = \frac{\alpha_{global}}{\alpha_{local}} \beta_{local}
\]

Letting \( T_{global} = T_{logical} \), we obtain the expression of the logical time as follows:

\[
T_{logical}(t) = \eta * T_{local} + \Delta T
\]  

In this case, skew acts as the relative logical clock rate and \( \Delta T \) the clock offset between the hardware clock and the logical clock. The logical clock is maintained as a software function and is only calculated on request based on a given hardware clock reading.

3.2 Clock drift management

The angular frequency of the hardware oscillator determines the rate at which the clock runs, the rate of a perfect clock should equal 1. However, all clocks are subject to clock drift; oscillator frequency will vary unpredictably due to various physical effects. Even though the frequency of a clock changes over time, it can be approximated with good accuracy by an oscillator with a fixed frequency.

The clock drifts shown in Figure 1 are an environment-related phenomenon and are hard to be modelled accurately. Even if we assume that the accurate drifts can be estimated by measuring the environment parameters in real-time, inconsistencies still remain in reading the corresponding values from drift-over-environment tables that may be perfectly known beforehand. As a matter of fact, clock drifts are not unique elements of the errors of a node’s local clock. If we sample X’s time over a very small period, the environmental factors will not change much; thus the drifts can be taken as constant and the curve in Figure 1 can be approached very closely by a sequence of linear regression functions. In each sampling period, we use certain methods to obtain time synchronisation. In order to obtain the accurate time, we can assume that the sampling time is high enough for us to take possession of time drifts as linear regression functions.

Figure 1 Relationship between a local time and the time source on TelosB motes (see online version for colours)

The node’s logical times are obtained using equation (4) and the relative clock rate is denoted as \( \eta \). With perfect clocks (no drift) the skew value will be 1. However, real clocks...
occasionally suffer from clock drift, as exhibited in Figure 1. In our protocol, a one-way sender–receiver synchronisation algorithm is adopted. The sender broadcasts its global time with the sending time-stamp as $T_{send}$. Once the receiver receives the message containing synchronisation information, the arriving time-stamp is saved as $T_{arrive}$ as depicted in Figure 2. The receiver keeps the pair data ($T_{send(i)}$ and $T_{arrive(i)}$) used for calculation. For the convenience of implementation, equation (1) can be transformed as follows:

$$T_{\text{logical}}(i) = \delta(i) + T_{\text{local}} + \beta_{\text{global}}$$

where $\delta(i) = \eta - 1$.

Using least squares for linear fitting, we represent the relative clock drift as:

$$Skew = \sum_{i=1}^{n} \left( T_{\text{offset(i)}} - \bar{T}_{\text{offset}} \right) \left( T_{\text{arrive(i)}} - \bar{T}_{\text{arrive}} \right)$$

$$\sum_{i=1}^{n} \left( T_{\text{arrive(i)}} - \bar{T}_{\text{arrive}} \right)^2$$

Thus the logical time can be obtained as follows:

$$T_{\text{logical}} = T_{\text{local}} + \bar{T}_{\text{offset}} + Skew \left( T_{\text{local}} - \bar{T}_{\text{arrive}} \right)$$

$$\bar{T}_{\text{offset}} = \frac{1}{N} \sum_{i=1}^{N} \left( T_{\text{offset(i)}} - \bar{T}_{\text{offset}} \right) \% \delta$$

$$\bar{T}_{\text{arrive}} = \frac{1}{N} \sum_{i=1}^{N} \left( T_{\text{arrive(i)}} - \bar{T}_{\text{arrive}} \right) \% \delta$$

Figure 2 Data packet transmitted over the radio channel

<table>
<thead>
<tr>
<th>sender:</th>
<th>preamble</th>
<th>sync</th>
<th>data</th>
<th>crc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Propagation delay</td>
<td></td>
<td>$T_{\text{send}}$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>receiver:</th>
<th>preamble</th>
<th>sync</th>
<th>data</th>
<th>crc</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_{\text{arrive}}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3.3 Error compensation

For the clocks used in networks, we will focus entirely on the present and future clock times. The arithmetic of linear regression takes charge of the time segment of collecting data for calculation; the parameters acquired gives a good approach for the past time but additional errors may occur at the present time or in the future. As such, the prediction of the time errors mentioned previously is necessary; and thus, a compensation algorithm is implemented to compensate for the prediction error.

In this section, we describe our clock synchronisation algorithm. The basic idea of the algorithm is to use linear regression to achieve time drift between nodes, and an error prediction approach is used to obtain one-step look-ahead prediction in the compensation mechanism.

Error prediction is the basic technology of our compensation algorithm and many approaches have been proposed for this prediction.

- The Exponentially Weighted Moving Average (EWMA) estimator is very simple and memory efficient, requiring constant storage of the old estimate for any kind of history tuning. EWMA uses a linear combination of historic data, which is weighted exponentially. As it is reactive to small shifts, EWMA is often used as an agile detector in many statistical process control applications.

- The moving average estimator is another widely used simple estimator. The algorithm works as follows. Let $n$ be the tuning parameter for the maximum size of a history window of bits. At any given event, append the history data, $h(i) (i=1, 2, \ldots s)$, with the number of $s$. Thus the prediction of the current event is $\hat{p} = \sum_{i=1}^{s} h(i) / s$.

The moving average estimator applies the same weight to all packets within the sliding window. We improve the algorithm by applying a weighting function which places a larger weight on more recent samples so that the estimation can be more adaptive to temporal changes. The basic algorithm works the same as the moving average but it adds the time weighted function $w$. The tuning parameters for this estimator are $n$ and $w$. In our study, we use only one weighting function, $w$, and only one tune $n$. While $w$ is not a perfect function, it serves the purpose of observing the effect on weighting.

The prediction algorithm works as follows. Let $h$ be the sequence of the history data under the sliding window $s$ and let $w$ be the weight of each data with a sequence of length $s$. We apply weight = 1 for the most recent $\lfloor s/D \rfloor$ packets in $h$ and decrease the weight for the most recent $\lfloor s/D \rfloor$ samples linearly from 1 to $1/\lfloor s/D \rfloor$. Here $D$ in $(D \in (0, s))$ implies the number for dividing the difference of the weightiness of the history data for prediction. If $D = 1$, the prediction will be the same as for the moving average estimator. From the above analyses, the prediction can be denoted as:

$$\hat{p} = \frac{\sum_{i=1}^{s} w(i) \cdot h(i)}{\sum_{i=1}^{s} w(i)}$$

The implementation of this estimator uses the same sliding widow in its moving average algorithm. Figure 3(a) illustrates $\hat{p}(i)$ of moving average estimator and the weighting moving average estimator with $n = 5$ and $D = 3$. The weighting moving average algorithm performs closer to the real value than the moving average. A visual comparison of the weighting moving average with different sliding windows is shown in Figure 3(b). The estimator with window size $n = 10$ performs significantly worse than the estimator with $n = 3$. At a higher window size ($n = 10$), time history persists for a longer period with a lower relation to the estimated point. A quantitative depiction of the estimator is shown in Table 1. As can be observed from Table 1, the weighting moving average estimator with $n = 3$ and $D = 2$ perfectly estimates the prediction synchronised error.
Figure 3 Results for different estimators and different parameters (see online version for colours)

(a) Moving average estimation \((n = 5)\) and weighting moving average estimation \((n = 5, D = 3)\)

(b) Weighting moving average estimation \([n = 3, D = 2]\) and \([n = 10, D = 2]\)

Table 1

<table>
<thead>
<tr>
<th>Estimator</th>
<th>Correlation coefficient with real value</th>
<th>Prediction average error</th>
<th>Error variance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Moving average (n = 5)</td>
<td>0.530226</td>
<td>-0.01442</td>
<td>419.4299</td>
</tr>
<tr>
<td>Moving average (n = 5, D = 3)</td>
<td>0.636017</td>
<td>-0.006716</td>
<td>345.1439</td>
</tr>
<tr>
<td>Weighted moving average (n = 3, D = 2)</td>
<td>0.655677</td>
<td>0.003731</td>
<td>327.9981</td>
</tr>
<tr>
<td>Weighted moving average (n = 10, D = 2)</td>
<td>0.050147</td>
<td>0.01224</td>
<td>642.9380</td>
</tr>
</tbody>
</table>

The compensation algorithm will work once \(\hat{P}\) is derived and new time information has arrived. As shown in Figure 4, the black line acts as the time of node A and the red dashed line denotes the logical time of node B relative to TA. Once \(\hat{P}\) is compensated for after the calculation, the static error of the logical time in node B can be eliminated and the same process applied to the next period and so on.

Synchronised nodes make smooth linear compensations before the next prediction period. Although we cannot obtain perfect synchronisation, which has an ideal precision of zero, synchronised nodes can make use of infinite approximation from their time sources.

Figure 4 The compensation algorithm (see online version for colours)

4 Implementation

This section describes the implementation of our Compensation Time Synchronisation Protocol (CTSP) in the SIA2420 sensor nodes using the TinyOS operating system.

4.1 Target platform

The hardware platform used to implement the protocol is the SIA2420 sensor node from the Shenyang Institute of Automation in China. It features a TI MSP430 microcontroller with 10 kB RAM. The CC2420 radio module has been designed for low-power applications and offers data rates up to 250 kBaud by using a Direct Sequence Spread Spectrum (DSSS).

The MSP430 microcontroller has 10 built-in 16-bit timers, three of which are denoted as timer A and the other seven as timer B. The SIA2420 board is equipped with two different quartz oscillators (32 kHz and 8 MHz) which can be used as clock sources for the timers. Timer A is configured to operate at 1/8 of the oscillator frequency (8 MHz), leading to a clock frequency of 1 MHz. Since Timer A is powered by an external oscillator, it is also operational when the microcontroller is in low-power mode. We employ Timer 2 in Timer A to provide our system with a free-running 32-bit hardware clock which offers precision to one microsecond. This approach on the SIA2420 node offers better clock granularity than more recent hardware platforms which lack a high frequency external oscillator, as shown in Table 2.
Table 2: Comparison of clock sources for common sensor network hardware platforms

<table>
<thead>
<tr>
<th>Platform</th>
<th>CPU clock</th>
<th>Quartz crystal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mica2</td>
<td>8 MHz</td>
<td>32 kHz, 7.37 MHz</td>
</tr>
<tr>
<td>MICAz</td>
<td>8 MHz</td>
<td>32 kHz, 7.37 MHz</td>
</tr>
<tr>
<td>TinyNode</td>
<td>8 MHz</td>
<td>32 kHz</td>
</tr>
<tr>
<td>Tmote Sky</td>
<td>8 MHz</td>
<td>32 kHz</td>
</tr>
<tr>
<td>SIA2420</td>
<td>8 MHz</td>
<td>32 kHz, 7.37 MHz</td>
</tr>
</tbody>
</table>

4.2 TinyOS implementation

The implementation of CTSP on the SIA2420 platform makes use of TinyOS 2.1. The protocol implementation provides time synchronisation as a service for an application running on the mote. The architecture of the time synchronisation component and its relation to other system components is shown in Figure 5.

Figure 5: Architecture of time synchronisation service and its integration within the hardware and software platforms (see online version for colours)

The root node first broadcasts a synchronisation message with its local clock acting as the global time of the network. Nodes that can overhear the message note the receiving time-stamp from the hardware clock model. The CTSP module of the root and synchronised nodes periodically broadcasts a synchronisation beacon containing the sending time-stamp. Each node overhearing messages sorts them by the choice of time source managed by the CTSP module and the run compensation mechanism is disposed of by CTSP module. Then the nodes that receive messages update their current information for synchronisation and renewal of the current offset between the hardware and logical time and the relative rate between the local and logical clocks according to equations (2) and (3). Error prediction is employed as in equation (4) and the compensation algorithm is proposed to obtain reliable synchronisation as in Figure 4.

5 Testbed experiments

We evaluated the implementation of CTSP by experiments on a testbed which consists of 30 SIA2420 sensor nodes. Experiments with the identical setup were also performed for FTSP and for TPSN, which is the basic time synchronisation protocol in TinyOS. A 5 × 7 network testing scenario, as shown in Figure 6, is employed for the experiment. Two other motes were used in the experiment: the reference broadcaster and the base station. The reference broadcaster queried the global time from all nodes in the network once every 30 seconds, and the base station collected the responses to the query from all the nodes. The topology of the 35 node network was enforced in software and, therefore, all the nodes were placed within the radio ranges of the reference broadcaster and the base station. In this way, the base station and the broadcaster were able to communicate directly with all 35 nodes and no multi-hop routing was necessary.

Figure 6: The test scenario of time synchronisation test

5.1 Experiment results

We tested the protocol focusing on the precision after compensation with the test scenario in Figure 6. Synchronisation messages were broadcasted every 30 seconds and nodes within the range of radio could overhear these messages for synchronisation. To verify the compensation mechanism in CTSP, we executed experiments with and without the compensation mechanism; the results are shown in Figure 7. We measured the average neighbour error without compensation of 3.8 us and 1.3 us for the algorithm with compensation. The variance of these two methods was 1.15 and 0.18, respectively. Thus the mechanism without compensation experiences rough time errors and great variance.

The resultant distribution of errors was collected from all one-day trials to produce Figure 8, which shows the distribution of errors in the accuracy of time synchronisation for each of the three algorithms. The error distributions of all techniques appear to be roughly Gaussian. Table 3 presents the mean in microseconds and variances as obtained by using cftool in MATLAB to analyse the distribution of time errors. CTSP exhibits better performance in both the figure and the table for all algorithms. The compensation algorithm decreased the fluctuation caused by clock drift and centralised error distribution, as shown in Figure 8. This is important for high precision applications, such as location and data fusion.
Figure 7  Comparison of network average errors with and without compensation (see online version for colours)

(a) Network average error without compensation

(b) Network average error with compensation of CTSP

Figure 8  Comparison of time error distribution with different time synchronisation protocols (see online version for colours)

Table 3  Experiment results for the estimators and parameters

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Mean</th>
<th>Variance</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTSP</td>
<td>1.101</td>
<td>0.398</td>
</tr>
<tr>
<td>FTSP</td>
<td>3.652</td>
<td>1.049</td>
</tr>
<tr>
<td>TPSN</td>
<td>11.15</td>
<td>3.756</td>
</tr>
</tbody>
</table>

6 Conclusion and future improvement

We have described the error compensation algorithm for the time synchronisation protocol in WSNs. The time synchronisation protocol with this algorithm was implemented on SIA2420 platforms running TinyOS. The average precision is no more than 5 μs and robustness to environment changes. This performance is markedly better than that of other existing time synchronisation approaches on the same platform.

The algorithm was tested and its performance was verified in a real world application. This is important because the service had to operate not in isolation but as part of a complex application where resource constraints, as well as intended and unintended interactions between components, can and usually do cause undesirable effects. Moreover, the system operated in the field with temperature and other diverse radio environmental changes.

Several researches thrusts remain in this area. The first is that greater accuracy error prediction should be introduced to stabilise the time synchronisation. Also, compensation for multi-hop nodes must be studied in future research.

Acknowledgements

This work is supported by National Natural Science Foundation of China 60804067 and National Excellent Young Leader Foundation under grant number 60725312.

References


Error compensation algorithm in WSNs synchronisation


