Simsync: A Time Synchronization Simulator for Sensor Networks[1)]

XU Chao-Nong[1,2,3] ZHAO Lei[1,3] XU Yong-Jun[1,3] LI Xiao-Wei[1,3]

[1](*Advanced Test Technology Lab, Institute of Computing Technology, Chinese Academy of Sciences, Beijing* 100080)

[2](*Department of Computer Science, Hefei University of Technology, Hefei* 230009)

[3](*Graduate School of Chinese Academy of Sciences, Beijing* 100080)

(E-mail: xu_chaonong@ict.ac.cn, xyj@ict.ac.cn, lxw@ict.ac.cn)

Abstract Time synchronization is a critical middleware service of wireless sensor networks. Researchers have already proposed some time synchronization algorithms. However, due to the demands for various synchronization precision, existing time synchronization algorithms often need to be adapted. So it is necessary to evaluate these adapted algorithms before use. Software simulation is a valid and quick way to do it. In this paper, we present a time synchronization simulator, Simsync, for wireless sensor networks. We decompose the packet delay into 6 delay components and model them separately. The frequency of crystal oscillator is modeled as Gaussian. To testify its effectiveness, we simulate the reference broadcast synchronization algorithm (RBS) and the timing-sync synchronization algorithm (TPSN) on Simsync. Simulated results are also presented and analyzed.

Key words Time synchronization, sensor networks, simulator

1 Introduction

Recently, research attention has been drawn towards wireless sensor networks (WSN). Thousands or even millions of small, low-power nodes are deployed in target environment, interacting with the environment and communicating with each other frequently. Consumers and researchers believe that WSN is expected to affect many aspects of our lives.

Time synchronization is one of the basic middleware services of WSN[1]. The aim of it is to keep the time of all nodes in network synchronized. It can be used in time-stamping sensor event, data fusion, time-division multiplexing access to the shared wireless media (TDMA), localization, energy-saving sleep mode, distributed logging, *etc*[2].

Most nodes in sensor network use crystal oscillator to do timekeeping. However, the frequency of crystal oscillator is influenced by factors such as temperature, pressure, manufacture technics, voltage, etc[3]. It is impossible to keep the time of all nodes synchronized without time synchronization algorithm.

Some time synchronization algorithms for WSN have been proposed over the past few years[2,4∼7]. However, sensor network is highly application oriented. As for time synchronization, it is even more sensitive to application if time is stamped on application layer. So it is necessary to evaluate the performance of time synchronization algorithm quickly so as to verify if it is fit for specific application.

Simsync is a time synchronization simulator for wireless sensor network. Based on the time character of Mica2[8,9] which is a popular testbed in WSN, it simulates the execution of time synchronization algorithm and enables fine-grained simulation on network time. We can easily develop, test and debug time synchronization algorithm with it.

Most existing simulators for WSN do not support simulation on time synchronization because it is hard to model packet delay and clock drift. Many researchers model them randomly to simulate their algorithms, so their simulated results are incomparable. Based on the time character of Mica2, we decompose packet delay into 6 delay components and model them separately. We also model the frequency of crystal oscillator as Gaussian so that clock drift can be setup. We also have setup a set of simulation parameters for Simsync and found them effective in simulation.

The organization of the paper is as follows. Section 2 introduces related works. Section 3 introduces the model of the packet delay and the crystal oscillator′s frequency. Section 4 introduces the structure of Simsync. We focus on state diagram of node and the time synchronization table which is the fundamental data structure in Simsync. Section 5 presents and analyzes the simulation result of RBS and TPSN. The last section is future work and the conclusion of this paper.

1) Supported in part by National Basic Research Program of P. R. China(2005CB321604), and in part by National Natural Science Foundation of P. R. China (90207002)

Received December 21, 2005; in revised form March 19, 2006

2 Related works

Emstar[10] provides a flexible environment for transitioning between simulation and deployment for iPAQ-class sensor nodes running Linux. To deal with the difficulty of modeling RF propagation for short-range, low-power radios in complex environment, Emstar provides the ceiling array where each virtual node is bridged to a real-world one for networking. As far as time synchronization is concerned, the packet delay modeled in Emstar is the sum of two parts: one is the packet sending time which is the result of the bit number of the packet divided by transmission baud rate, the other is a random time varying between $-1/2T_b$ and $+1/2T_b$ where T_b is the time for sending a single binary bit.

TOSSIM[11,12] is a discrete event simulator for TinyOS based sensor networks. User can compile TinyOS application into TOSSIM framework and run it on PC. Regretfully, TOSSIM focuses on simulating TinyOS and its execution rather than simulating the real world. It does not model packet delay, nor does it model clock drift. So TOSSIM can not be used to evaluate time synchronization algorithm.

Omnet++[13] is a discrete event simulator developed by Andras Varga. It supports modeling wireless channels and packet delay by user with c++. In [14], the writer uses it to simulate his algorithm by modeling the sender delay and receiver delay as a Gaussian variable with a mean of 100μs and a standard deviation of 11μs. Reference [15] also uses it to do simulation but the writer has not introduced its simulation model in detail.

Avrora[16] is a speed-up version of ATEMU[17]. Both of them can simulate in instruction level. ATEMU is a simulator not for sensor network but for AVR microcontroller. Avrora uses two strategies to speed up simulation and is for as much as 20 times faster than ATEMU in a network of up to 10,000 nodes. Both Avrora and ATEMU have not modeled clock drift.

3 Time model

To simulate time synchronization algorithms for WSN, it is necessary to setup the simulation model for node′s time. At present, many time synchronization algorithms running on Mica2 are based on packet exchanging[2,4∼7]. So modeling the packet delay is essential.

As mentioned before, time asynchrony among nodes is due to the difference of crystal oscillators′ frequencies. The time of node is directly influenced by the frequency of its crystal oscillator. So modeling the frequency of crystal oscillator is also essential.

3.1 Model of packet delay

The biggest enemy of precise time synchronization of sensor network is non-determinism. Reference [5] decomposes the packet delay into the following 6 components.

Send time: the delay which is spent in assembling a packet and delivering the packet to MAC layer in sender. It depends on the system call overhead of the operation system and the load of processor. It is nondeterministic.

Access time: the delay incurred by waiting for access to the wireless channel. It is the least deterministic part of packet delay.

Transmission time: the delay it takes for a sender to transmit the packet bit by bit at physical layer. It depends on the length of the packet and the transmission baud rate.

Propagation time: the delay it takes for one binary bit in packet to travel the wireless link from the sender to the receiver. It is deterministic and depends on the distance between the sender and the receiver.

Reception time: the delay it takes for the receiver to receive the packet. Same as transmission time, it depends on the length of packet and the transmission baud rate. It may partly overlay with transmission time.

Receive time: the delay of processing the incoming packet and delivering it to the application layer in the receiver. Its character is similar to that of sending time.

Fig. 1 is the outline of the packet delay. We now look into the components of the packet delay in Mica2 in detail. We focus on how the first bit in a packet is delivered. At first, the application layer in the sender assembles the packet and stamps the time, after T_{send} (send time), the packet is delivered to MAC layer. After T_{access} (access time), the sender gains access to wireless channel, then the first bit is transmitted to radio chip through its IO pin (the next bit will be transmitted after T_{perbit} which is the reciprocal of the transmission baud rate). After $T_{encoding}$, which is called the encoding time, the radio chip encodes and transforms the first bit into electromagnetic waves. When the electromagnetic waves propagate to the receiver after $T_{propagation}$ (propagation time), the radio chip of the receiver receives the electromagnetic waves. After $T_{decoding}$ which is called the decoding time, it transforms and decodes the electromagnetic waves into a binary bit and transmits the bit to microcontroller through IO pin. Both $T_{encoding}$ and $T_{decoding}$ are deterministic and depend on the character of radio chip. When the first byte is received by hardware of receiver, it interrupts the microcontroller to request for processing. After $T_{interrupt}$ (interrupt time of microcontroller), the interrupt server program is executed and the first byte is saved into memory. When all bytes in the packet are received, they will be delivered from MAC layer to application layer in the receiver after $T_{receive}$ (receive time). The application layer on the receiver stamps the time to indicate that it has received a packet.

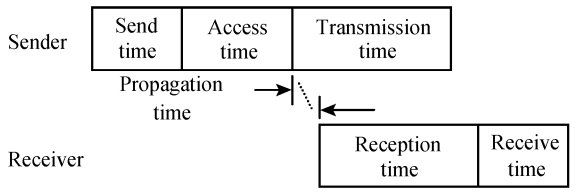

Fig. 1 Components of packet delay

So the delay of the first byte is

$$T_{first} = T_{send} + T_{access} + T_{encoding} + T_{decoding} + T_{propagation} + T_{\text{int}\ errupt} + T_{receive}$$

If all bytes in the packet are received correctly, the following equation comes into existence.

$$T_{perbit} = MAX(T_{perbit}, T_{encoding}, T_{decoding}, T_{\text{int}\ errupt}, T_{propagation})$$

If L stands for the bit number of the packet when it is in MAC layer, based on the pipeline theory[18], the delay of a packet is

$$T = T_{first} + (L-1)T_{perbit} \tag{1}$$

Those equations seem to be perfect, however, it is hard to know the values of those delay components because some of them are greatly influenced by factors such as the burden of microprocessor, the chance of wireless collision and the transmission baud rate. We find that in many experiments of time synchronization algorithms, researchers often test their time synchronization algorithms based on TinyOS and Mica2. So the values of those delay components must be consistent with the time character of TinyOS and Mica2 for compatibility.

After many experiments, we find a set of reasonable values for these delay components: T_{send} and $T_{receive}$ are modeled as a Gaussian random variable from 45μs to 55μs. T_{perbit} is 52μs because the default baud rate of Mica2 is 19.2kbps. Both $T_{encoding}$ and $T_{decoding}$ are 100μs. $T_{propagation}$ can be computed according to the distance between sender and receiver. T_{access} is the interval between the time when the MAC layer receives the packet from its upper layer and the time when the sender gains access to the wireless channel. In Simsync, T_{access} is the sum of the packet transmission time and the interval from now to the time when wireless channel is idle. $T_{interrupt}$ is modeled as a Gaussian random variable from 48μs to 75μs.

We need to point out that the values of these delay components have already taken wireless collision and the overhead of TinyOS into account. Simsync also provides interface for user to configure them himself.

3.2 Model of crystal oscillator frequency

Any crystal oscillator ticks at a slightly different rate even if it is claimed to be of the same frequency, because factors such as temperature, pressure, manufacture technics, voltage, *etc* can influence it. The synchronization precision will decay more seriously as more time between the latest synchronization operation and the event of interest elapses. The frequency of crystal oscillator is accurate on the order of from 1ppm to 100ppm, that is, time between two nodes may drift 0~100 microseconds per second if the nominal frequency of crystal oscillator is 1MHZ. In our model of crystal oscillator frequency, we model the instantaneous frequency of a crystal oscillator as a Gaussian variable (the mean is its nominal frequency, while the deviation is from frequency/10^{-6} to frequency/10^{-4} according to the type of crystal oscillator). We assume that the instantaneous frequency is stable during a short period which is the interval of the basic simulation step in Simsync. It is set as 1μs in Simsync.

The model of crystal oscillator frequency is used to transform between the local interval and the global interval. Their difference is due to the different timescale. For a given time interval of a node, the global interval is based on the nominal frequency of its crystal oscillator while the local interval is based on its real frequency. The local interval is equal to the global interval only if the nominal frequency is equal to the real frequency of the node′s crystal oscillator. Their relationship is as follows.

$$\frac{local\ interval}{global\ interval} = \frac{real\ frequency}{nominal\ frequency} \tag{2}$$

According to (2), based on the global interval, the local intervals of different nodes can be transformed into each other.

4 Structure of Simsync

In this section, we first setup the state diagram of sensor node. Its state will transform during the transmission of packet. Then we introduce the time synchronization table, which is the most fundamental data structure in Simsync for state transformation. At last, we introduce the program structure of Simsync.

4.1 The states of node

Fig. 2 is the status diagram of both sender and receiver.

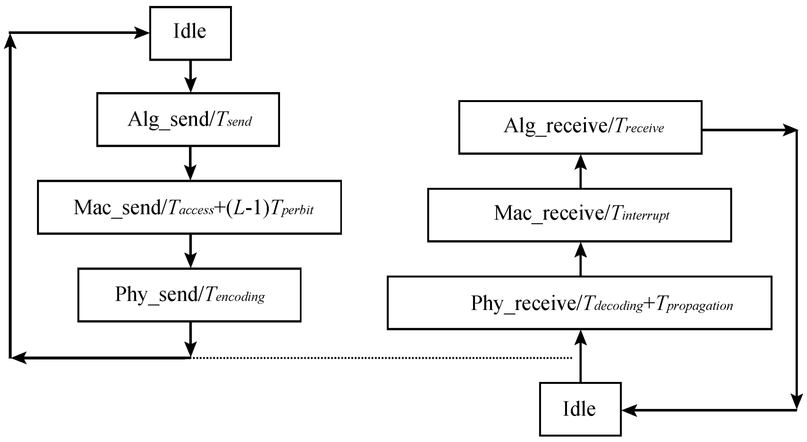
Idle
Alg_send/T_{send}
Mac_send/T_{access}+(L-1)T_{perbit}
Phy_send/$T_{encoding}$
Alg_receive/$T_{receive}$
Mac_receive/$T_{interrupt}$
Phy_receive/$T_{decoding}$+$T_{propagation}$
Idle

Fig. 2 Status diagram of sender and receiver

To serially simulate the transmission of a packet, we divide the state set of a sender into the following 4 states:

idle: the node has no transmission task or has already finished transmitting a packet.

alg_send: when the node has prepared a packet to be sent, the node will enter into alg_send state. As we know, the state will persist for T_{send}.

mac_send: when the packet has been delivered to MAC layer, it will enter into mac_send state; the state will persist for $T_{access} + (L-1)T_{perbit}$, where L is the bit number of the packet in MAC layer.

phy_send: when the last bit in the message is delivered to physical layer, it will enter into phy_send state, the state will persist for Tencoding.

We also divide the state set of a receiver into the following 4 states:

idle: the node has no receiving event or has already finished receiving and stamping time on the packet.

phy_receive: when the last bit of the packet propagates in the air, every neighbor node of the sender will enter phy_recieve state; the state will persist for $T_{propagation} + T_{decoding}$.

mac_receive: when the last bit of the message is received by hardware and the last byte will be delivered to MAC layer, the node will enter into mac_receive state; the state will persist for $T_{interrupt}$.

alg_receive: when a packet is received by MAC layer and delivered to application layer, the node will enter into alg_receive state; the state will persist for $T_{receive}$.

It can be easily seen that the sum of the all states$'$ persisted time in both sender and receiver is the delay of the packet.

The reason why the packet delay is decomposed into so many delay components is that the architecture can support many techniques in time synchronization. For example, MAC layer time-stamping can be easily implemented by setting T_{access}, T_{send} and $T_{receive}$ as 0.

4.2 Time synchronization table

Time synchronization table is derived from the event queue in TinyOS. From the view of its data structure, it is actually an ordered queue. Each cell in the time synchronization table represents an event which will synchronize all nodes in the network. All events in the time synchronization table are ordered based on their occurring time. Fig. 3 illustrates the structure of a cell in the time synchronization table. We have to point out that the Advanced_time in the cell is a global interval. When a cell is taken out from the table, on the one hand, the state of Influenced_node will be changed into Next_state, on the other hand, since the Advanced_time is a global interval, it has to be transformed into the local interval for every node according to (2), then every node in the network will advance its time for its corresponding local interval.

Influenced_node	Advanced_time	Next_state

Fig. 3 Structure of a cell in the time synchronization table

When a node needs to insert one cell into the table, first, it transforms its local interval into global interval which is notated as Advanced_time. Then, it needs to find a position in the time synchronization table based on the value of Advanced_time. At last, it modifies the value of Advanced_time and inserts the cell into the table. If we notate the cell as CellA, and the i$'$th cell in the table is notated as table[i]. The procedure is as follows.

```
$i = 0;$        $Tpersist = \&(cellA.Advanced\_time);$
$while((table[i].Advanced\_time < Tpersist)\&\&(i <= number\ of\ cell\ in\ queue))$
    $\{$  $Tpersist = Tpersist - cell[i].Advanced\_time;$
          $i++;\}$
    $Insert\ CellA\ into\ queue[i];$   $\}$
```

4.3 Main program structure of Simsync

The main program structure of Simsync is table-driven. It is as follows:

```
$while(!empty(time\ synchronization\ table))$
    $\{$  $cell = outtable();$
          $process(cell);$
          $Advancetime(cell.advanced\_time);$   $\}$
```

5 Experiment

Since there are so many time synchronization algorithms at present, we have to decide which one is suitable for verifying the effectiveness of Simsync. From the view of the synchronization scheme, there are two categories of the time synchronization algorithms which run on Mica2. They are the sender-receiver synchronization algorithms and the receiver-receiver synchronization algorithms. The representative of the receiver-receiver synchronization algorithms is RBS algorithm, and the representative of the sender-receiver synchronization algorithms is TPSN algorithm. Many other algorithms are derived from them. So we decide to simulate RBS algorithm and TPSN algorithm, then compare the simulated results with those reported by other researchers.

We first realize RBS algorithm. Three nodes are deployed in a single-hop network. The frequency of every node$'$s crystal oscillator is a random number from 7.37353728MHZ to 7.37206272MHZ, that is, the stability of the crystal oscillator is from -100ppm to 100ppm. One node is appointed as the reference node and the other two nodes are the receiver of the reference broadcast messages. We simulate it for 50 minutes. The synchronization cycle is 10 seconds.

At the beginning of every synchronization cycle, we inquire and record the time of every node in sequence. At the end of simulation, all these time is exported as a file which is used for further analysis. So there are 300 time items for each node.

Based on the exported file, we work out the difference between the two receivers for every time item. The difference is the synchronization error between them. If we think of the synchronization error as a random variable, there are 300 samples for it since there are 300 time items for every node.

Table 1 lists the statistics of RBS algorithm from 4 different sources, where μ is the mean of the synchronization error and σ is standard deviation of the synchronization error. Their units are microsecond(μs). Δ is the percentage of time error which is less than or equal to the average error.

According to Table 1, we can see that the result reported in [5] has great a gap with result reported in [4]. We think that it is mainly due to the different test platforms. In [4], the test platform is strongARM-based Compaq IPAQs with Lucent technologies 11Mbit 802.11 wireless Ethernet adapters while test platform of [5] and [19] and the simulated object of Simsync are all Berkley mote.

The result reported in [5] has some difference with the result reported in [19]. We think it is normal considering the delicacy of time. Even the slight different realization of the same algorithm may influence the experiment results in some degree.

Table 1 Statistics of RBS algorithm

Paper	Interval	$\mu(\mu$s)	$\sigma(\mu$s)	Max error(μs)	Min error(μs)	Δ
[4]	8s	6.29	6.45	Not available	Not available	Not available
[5]	10s	29.13	Not available	93	0	53%
[19]	10s	20.3765	22.4728	Not available	Not available	Not available
SimSync	10s	26.9167	20.8943	103.4312	0.1346	59.9%

We can also see the simulated result on Simsync is between the results reported in [5] and [19]. Their standard deviations are also very alike.

We also realize TPSN algorithm. There are only two nodes deployed in a single-hop network. All other parameters are same with that of RBS simulation. At first, we do not employ the technique of MAC-layer time stamping. Its result is shown in the first line of Table 2. Then, we employ the technique of MAC-layer time stamping. Its result is shown in the second line of Table 2. The third line of Table 2 is from [5]. It also employs the technique of MAC-layer time stamping. It can be easily seen that the technique of MAC-layer time stamping indeed improves the synchronization precision of TPSN algorithm.

[7] states that the byte alignment time[7] is compensated in [5] although the writer has not mentioned it explicitly. So the synchronization error in [5] must be smaller than our simulated results. The result advocates the effectiveness of Simsync.

Table 2 Statistics of TPSN algorithm

Paper	Interval	$\mu(\mu$s)	$\sigma(\mu$s)	Max error(μs)	Min error(μs)	Δ
SimSync	10s	42.3319	34.0716	176.9592	0.0075	63.7%
SimSync	10s	30.6976	23.3511	120.3689	0.0087	59.3%
[5]	10s	16.9	Not available	44	0	64%

6 Conclusion and future work

We have implemented a time synchronization simulator for wireless sensor networks. The packet delay from sender to receiver is decomposed into 6 delay components. All those delay components are modeled separately. Furthermore, the frequency of crystal oscillator is modeled as Gaussian. With the data structure named time synchronization table, it enables fine-grained sequential simulation on network time. Users can use it to develop, debug and evaluate the time synchronization algorithm of their own. To verify its effectiveness, we realize RBS algorithm and TPSN algorithm based on it and get very close experiment result with the result reported in other three papers. All results advocate the effectiveness of Simsync.

In our future work, we will measure the packet delay and the frequency of crystal oscillator on Mica2, and set up a more accurate model to make simulations more realistic.

References

1 Elson J, Römer K. Wireless sensor networks: A new regime for time synchronization. In: Proceedings of the First Workshop on Hot Topics in Networks (HotNets-I). Princeton, New Jersey: 2002

2 Sichitiu M L, Veerarittiphan C. Simple, accurate time synchronization for wireless sensor networks. In: Proceedings of IEEE Wireless Communication and Networking Conference (WCNC 2003). New Orleans, LA: IEEE Press, 2003. 16∼20

3 John R Vig. Introduction to quartz frequency standards. Technical Report SLCET-TR-92-1, Army Research Laboratory, Electronics and Power Sources Directorate, 1992. [Online], available: http://www.ieee-uffc.org/freqcontrol/quartz/vig/vigtoc.htm. March 19, 2006

4 Elson J, Girod L, Estrin D. Fine-grained network time synchronization using reference broadcasts. In: Proceedings of the 5th Symposium on Operatiation System Design and Implementation (OSDI2002). Boston, MA, 2002. 147∼163

5 Ganeriwal S, Kumar R, Srivastava M. Timing-sync protocol for sensor networks. In: Proceedings of the 1st ACM Conference on Embedded Networked Sensor Systems (SENSYS 2003). Los Angeles, CA: ACM Press, 2003. 138∼149

6 Römer K. Time synchronization in *ad hoc* networks. In: Proceedings of the 2nd ACM International Symposium on Mobile *ad hoc* Networking & Computing (MobiHoc 01). Long Beach, CA: ACM Press, 2001. 173∼182

7 Maroti M, Kusy B, Simon G, Ledeczi A. The flooding time synchronization protocol. In: Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems, Baltimore, MD: ACM Press, 2004. 39∼49

8 Mica2 and Mica2Dot[Online], available: http://www.xbow.com/Products/Wireless_Sensor_Networks.htm. March 19, 2006

9 Hill J, Culler D. Mica: A wireless platform for deeply embedded networks. *IEEE Micro Archive*, 2002, **22**(6): 12∼24

10 Elson J, Bien S, Busek N, Bychkovskiy V, Cerpa A, Ganesan D, Girod L, Greenstein B, Schoellhammer T, Stathopoulos T, Estrin D. EmStar: An environment for developing wireless embedded systems software. Center for Embedded Networked Sensing (CENS) Technical Report, CENS-TR-9, 2003

11 Levis P, Lee N, Welsh M, Culler D. TOSSIM: Accurate and scalable simulation of entire TinyOS applications. In: Proceedings of the 1st ACM Conference on Embedded Networked Sensor Systems (SenSys′03). Los Angeles: ACM Press, 2003. 126∼137

12 TOSSIM: A Simulator for TinyOS Networks. User′s Manual, in TinyOS documentation

13 Varga A. The OMNeT++ discrete event simulation system. In: Proceedings of the European Multiconference (ESM′2001). Prague, Czech Republic, 2001. 319∼324

14 Ye Q, Zhang Y C, Cheng L. A Study on the optimal time synchronization accuracy in wireless sensor networks. *Computer Networks*, 2005, **48**(4): 549∼566

15 Greunen J V, Rabaey J. Lightweight time synchronization for sensor networks. In: Proceedings of 2nd ACM International Workshop on Wireless Sensor Networks and Applications (WSNA). San Diego: ACM Press, 2003. 11∼19

16 Titzer B, Lee D, Palsberg J. Avrora: Scalable sensor network simulation with precise timing. In: Proceedings of the 4th International Conference on Information Processing in Sensor Networks (IPSN′05). Los Angeles, 2005. 477∼482

17 Polley J, Blazakis D, McGee J, Rusk D, Baras J S, Karir M. ATEMU: A fine-grained sensor network simulator. In: Proceedings of the 1st IEEE Communications Society Conference on Sensor and *Ad Hoc* Communications and Networks (SECON′04). Santa Clara, CA: IEEE Press, 2004. 145∼152

18 Hennessy J L, Patterson D A. Computer Architecture: A Quantitative Approach. 2nd edition, San Mateo, CA: Morgan Kaufmann Pub, 1995

19 Dai H, Han R. TSync: A lightweight bidirectional time synchronization service for wireless sensor networks. *Mobile Computing and Communications Review*, 2004, **8**(1): 125∼139

Xu Chao-Nong Ph. D. candidate at Institute of Computing Technology, Chinese Academy of Sciences. His research interests include wireless sensor networks and embedded system.

ZHAO Lei Ph. D. candidate at Institute of Computing Technology, Chinese Academy of Sciences. His research interests include wireless sensor network and VLSI design.

XU Yong-Jun Ph. D. candidate at Institute of Computing Technology, Chinese Academy of Sciences. His research interests include wireless sensor networks and low power system.

LI Xiao-Wei Professor at Institute of Computing Technology, Chinese Academy of Sciences. His research interests include VLSI/SOC testing and embedded system.