Wireless Personal Communications (2007) 43:185–200
DOI 10.1007/s11277-006-9217-4

A Clock Synchronization Algorithm for Multihop Wireless Ad Hoc Networks

JANG-PING SHEU[1], CHIH-MIN CHAO[2], WEI-KAI HU[1] and CHING-WEN SUN[1]

[1]*Department of Computer Science and Information Engineering, National Central University, Jhongli 32001, Taiwan*

[2]*Department of Computer Science and Engineering, National Taiwan Ocean University, Keelung 20224, Taiwan*

E-mail: cmchao@axp1.csie.ncu.edu.tw

Abstract. In multihop wireless ad hoc networks, it is important that all mobile hosts are synchronized. Synchronization is necessary for power management and for frequency hopping spread spectrum (FHSS) operations. IEEE 802.11 standards specify a clock synchronization protocol but this protocol suffers from the scalability problem due to its inefficiency contention mechanism. In this paper, we propose an automatic self-time-correcting procedure (ASP) to achieve clock synchronization in a multihop environment. Our ASP has two features. First, a faster host has higher priority to send its timing information out than a slower one. Second, after collecting enough timing information, a slower host can synchronize to the faster one by self-correcting its timer periodically (which makes it becoming a faster host). Simulation results show that our ASP decreases 60% the average maximum clock drift as compared to the IEEE 802.11 and reduces 99% the number of asynchronism in a large-scale multihop wireless ad hoc networks.

Keywords: Clock synchronization, IEEE 802.11, multihop wireless ad hoc networks

1. Introduction

A wireless mobile ad hoc network (MANET) is formed by a cluster of mobile hosts without any pre-designed infrastructure of the base stations. In MANETs, it is important that all mobile hosts synchronize to a common clock. In frequency hopping spread spectrum (FHSS), synchronization is required to assure all mobile hosts hopping at the same time. Synchronization is also required to perform power management for both FHSS and direct sequence spread spectrum (DSSS). Without such clock synchronization, mobile hosts may not wake up at the same time and thus the power management operation may not work well [1]. A distributed *timing synchronization function* (*TSF*) is specified in IEEE 802.11 WLAN standard [2] to fulfill clock synchronization in a MANET. In this synchronization mechanism, each mobile host is responsible for exchanging timing information through the periodic beacon transmissions. A host synchronizes its clock according to the timestamp in the beacon if the received time is later than its own.

As the number of hosts increases, the transmission contentions uprise accordingly. As a result, the *scalability problem* occurs (performance analysis can be found in Refs. [3, 4, 5]). A scalability problem is also induced by this IEEE 802.11 standard TSF due to the beacon contentions [6, 7]. For a large scale MANET, beacon frames can hardly be successfully transmitted and some hosts may not be able to synchronize with others. An *adaptive timing* *synchronization procedure (ATSP)* is proposed in Ref. [6] to solve this scalability problem. The basic idea behind ATSP is from the observation that, in the IEEE 802.11 TSF, only later (faster) timing synchronizes others. Thus, ATSP gives the fastest host (the host that has the fastest timing) the highest priority to transmit beacons (by allowing the fastest node contends to transmit beacon every beacon interval). On the contrary, slower hosts' beacon transmission frequencies are reduced. This reduction is achieved by allowing a slower node i contends to transmit beacon every $I_{\max}$ beacon intervals, if no faster timing value is received by node i during these $I_{\max}$ beacon intervals. When the system reaches a *stable state*, the fastest node has a very high probability of sending a beacon successfully while the other nodes have the lowest priority. ATSP successfully alleviate the scalability problem but in some cases, such as the fastest mobile host leaves, some mobile hosts' clocks may still differ from others for hundreds of microseconds. To overcome these problems, a revision of ATSP, which is called *Tiered Adaptive Timing Synchronization Procedure (TATSP)*, is proposed in Ref. [7]. The basic idea of TATSP is quickly identifying a small set of fastest nodes and giving them higher priority to transmit beacons. Both ATSP and TATSP achieve clock synchronization for mobile hosts and scale well in a single-hop MANET. However, these two algorithms fail to function well when applying to a multihop MANET. The most important task of time synchronization in a multihop MANET is to relay faster timing from faster nodes. Both ATSP and TATSP take a long time to finish this task. For example, in a four-hop linear topology A–B–C–D where A is the fastest host, hosts B and C are responsible of spreading the time information. For the ATSP protocol, host A has the highest priority while host B has the lowest priority (low priority nodes can contend to transmit every $I_{\max}$ beacon intervals). In such a case, host B will have less chance to transmit a beacon and thus hosts C and D can synchronize to host A after a long time. The TATSP protocol revises the ATSP by adopting two different $I_{\max}$ values. Similar long-convergence-time problem happens in TATSP.

In this paper, we propose a clock synchronization algorithm, which is called *automatic self-time-correcting procedure (ASP)*, to achieve clock synchronization among mobile hosts and hence to solve the scalability problem in a multihop MANET. Two tasks must be done to fulfill clock synchronization in a multihop MANET: to increase the successful transmission probability for faster hosts and to spread the faster timing information throughout the whole network. In ASP, the first task is achieved by increasing the beacon transmission priority of a host who has faster timing and by cutting down the priorities of the others. And then, when some slower hosts get enough information to accomplish synchronization by themselves, their beacon transmission priorities are increased to carry out the second task. By efficiently carrying out these two tasks, the ASP mitigates the clock asynchronism occurred in IEEE 802.11 networks.

The rest of this paper is organized as follows. We review the clock synchronization protocol specified in the IEEE 802.11 standard in Section 2. Section 3 describes the details of the proposed clock synchronization algorithm, the ASP. Simulation results are in Section 4. Conclusions are given in Section 5.

2. Clock Synchronization of IEEE 802.11

IEEE 802.11 standards [2] specify the mechanisms of clock synchronization and power management in *media access control* (MAC) layer. Each mobile host shall maintain a TSF timer with modulus 2^{64} counting in increments of microseconds (μs). Hosts are responsible to contend transmitting beacon frames periodically. The host, who wins the contention will send a beacon frame which contains the host's TSF timer associated with other parameters. Other hosts adopt the received timing information only when the TSF timer is faster than their own. Specifically, hosts are synchronized with others by the TSF value (timestamp) contained in the beacon frames. Each host's TSF timer is the summation of a variable *offset* and the host's clock. When a host receives a beacon and finds its own TSF timer is slower than the timestamp in the beacon, it will add the timing difference to its *offset*. The interval between beacon frames is defined as the aBeaconPeriod, which specifies the length of a beacon interval and is an identical value for all hosts in the MANET. In other words, time is divided into a series of beacon intervals which are exactly aBeaconPeriod time units apart. A host will execute the following steps at the beginning of each beacon interval to achieve beacon generation and clock synchronization:

1. Suspend the decrementing of the backoff timer for any non-beacon transmission.
2. Generate a random delay uniformly distributed in the range between zero and twice aCWmin $\times$ aSlotTime. (The values of aCWmin and aSlotTime are 15 and 50 μs for FHSS and are 31 and 20 μs for DSSS.)
3. Wait for the random delay timer.
4. Cancel the random delay timer if a beacon is received from another host before the timer has expired. If the clock information in the receiving beacon is later than its TSF timer, adopt the value.
5. Send a beacon with the TSF timing information if the random delay timer has expired and no beacon has arrived during the delay period.

All hosts participate beacon generation at the beginning of each beacon interval, which is defined as the *beacon generation window* as shown in Figure 1. This window comprises ($2 \times$ aCWmin $+ 1$) time slots and each station is scheduled to transmit a beacon at one of these slots.

3. Clock Synchronization in Multihop MANETs

As mentioned earlier, to achieve clock synchronization in multihop MANETs, we have to uprise the faster beacons' successful transmission probability and then spread this faster timing out. The former task is accomplished through dynamically adapting hosts' beacon transmission frequencies, according to their own clock oscillation frequency, such that a faster host obtains higher priority to transmit a beacon. Slower hosts, after receiving the fast beacon twice from the same host (with the same *Seq_No*, which will be described later), can calculate the exact difference between their clocks. Then, even without receiving the faster beacons, slower

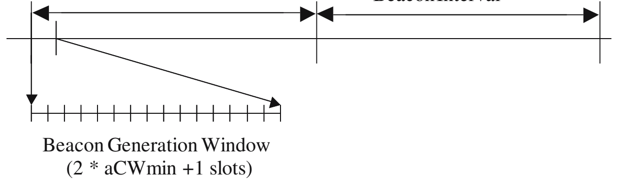

Figure 1. Beacon generation window.

hosts can automatically synchronize to the faster one. These hosts whose clocks have been corrected can then take the responsibility to spread the faster timing out. In the following, we first describe the mechanism to increase the faster beacons' successful transmission probability and then present the automatic self-time-correcting procedure.

3.1. Contention for Beacon Transmission

In a MANET, hosts must attend the contention for beacon transmission and each of them has equal probability to win the contention. To increase the number of successful faster beacon transmission, our basic idea is to reduce the probability of a slower host to transmit the beacons. An integer variable p_i is defined as the period, counted in the number of beacon intervals, for host i to transmit a beacon. For example, host i will try to transmit a beacon every 2 beacon intervals if $p_i = 2$. The setting of the p_i must reflect host i's clock such that a faster host can transmit beacon frequently. Also, p_i is related to the number of host i's neighbors, NB_i. The value p_i should be in proportion to NB_i since a large NB_i means a beacon is vulnerable to be collided with. We define p_i as follows.

$$p_i = \left\lfloor \left(\frac{\max(1, NB_i)}{\max(1, NL_i)} \right)^{\alpha} \right\rfloor, \quad \alpha \in N \tag{1}$$

where N is integer set and NL_i is the number of host i's neighbors who's TSF timer is equal to or slower than host i. The value of α is used to adjust the number of hosts to contend for the beacon transmission. A small α induces more hosts to transmit. For example, when $NB_i = 10$ and $\alpha = 1$, p_i will be one when NL_i is more than five. That is, the hosts belong to the faster half will attempt to transmit a beacon in every beacon interval. On the contrary, when $\alpha = 2$, p_i will be one when NL_i is more than seven, which means the number of hosts that can transmit in every beacon interval is reduced. The best value of α will be selected by simulations.

3.2. Automatic Self-time-correcting Procedure (ASP)

In this subsection, we describe the ASP in detail.

3.2.1. Self-time-correcting

ASP changes a mobile host's *offset* to achieve clock synchronization, which is the same as IEEE 802.11. In addition, a host running ASP tries to obtain the clock oscillation difference between itself and a faster host. With this information, a slower host can avoid asynchronism, even without receiving the faster beacons, by adding the oscillation difference to its *offset* periodically. A host i obtains the oscillation difference by comparing the difference of its TSF timers to the successively received faster beacons (from the same host). Specifically, we define *Pass_Time1* as the elapsed time that a host receives two successive beacons from the same host and *Pass_Time2* as the timestamps difference between these two beacons. The oscillation difference of these two hosts' clocks, *Diff*, equals to (*Pass_Time2* − *Pass_Time1*). To achieve self-time-correction, a slower host i should adjust its *offset* periodically. The interval, a_i, between each self-correction is defined as

$$a_i = \lfloor Pass\_time1/Diff \rfloor \tag{2}$$

That is, the slower host shall automatically increase one to its *offset* in every a_i microseconds to keep synchronized with the faster one.

Note that, strictly speaking, we define *Pass_Time2* by simply taking the elapsed time for successively two beacons from the same host is not correct. Consider the following situation: a host i receives the beacon transmitted by a faster host j in the first beacon interval. Next, a host k which is far from the host i transmit a faster beacon than host j does. Thus, host j updates its *offset*. In the next beacon interval, host j transmits another beacon carrying this modified TSF timer. Unaware of the beacon transmitted by host k, host i will estimate its clock oscillation to host j improperly. To overcome this problem, we add a 4-bit variable *Seq_No* in the beacon frame to keep track of the changes of TSF timer. Whenever the *offset* is updated because of a faster beacon, *Seq_No* is increased by one. The correct calculation of *Pass_Time2* shall be taken from two beacons that is transmitted by the same host with the same *Seq_No*. Note that it does no hurt for a slower host, say host A, to receive only one faster beacon with the same *Seq_No*. In such a case, host A still can synchronize its TSF timer to the fast beacon although it is incapable of doing self-correction.

To prevent the wraparound problem of Seq_No for TSF timer, the received timing information stored in each host will be abandoned after eight beacon intervals. Sometimes, in a multihop MANET, a host will receive more than one faster beacon in one beacon interval. Each of these beacons causes an *offset* update, which consume the Seq_No quickly. For example, three hosts A, B, and C are in line. Host B can communicate with A and C while A and C are hidden to each other. Suppose that host A has the fastest clock among the three, which is in turn followed by C and B. Now, host C transmits a beacon first. Host B will update its *offset* and Seq_No after recognizing this faster clock. Then, in the same beacon interval, host A (without receiving any beacon) also transmits its beacon and host B updates its *offset* and Seq_No again. This example illustrates the scenario that a host may update its Seq_No more than once in a beacon interval. That is why we cut the timeout threshold by half.

3.2.2. Data structure

Each mobile host maintains a *Neighbor_Table* to keep track of its neighbors and their TSF timers. Each host should also maintain a *Clock_Table* to record the information of neighbors who have faster TSF timers than its own. A Clock_Table consists of four fields: *MH_Id, Seq_No, Last_Recv_Clk,* and *Last_My_Clk*. MH_Id is the neighbor's identity, Seq_No is the sequence number of the neighbor's TSF timer, $Last\_Recv\_Clk$ is the last received beacon timestamp from the particular neighbor, and $Last\_My\_Clk$ is its own TSF timer value when $Last\_Recv\_Clk$ is received. With the information in Clock_Table and Eq. (2), mobile hosts are able to synchronize to faster clock automatically.

To make the ASP work properly, four variables are needed for each host i:

1. An integer variable Seq_No for TSF timer. This Seq_no is increased by 1 whenever the TSF timer is changed. The maximum value of Seq_No is 15. This value is included in the beacon frame.
2. An integer variable p_i which indicates the period a host i shall attempt to transmit a beacon. p_i is calculated by Eq. (1) using the information in its Neighbor_Table.
3. A counter c_i which counts for the number of beacon intervals that have elapsed since the host i attempt to transmit a beacon last. Initially, c_i is set to zero. When $c_i = p_i$, mobile host i shall transmit a beacon and c_i is reset to zero.

4. An interval a_i which is calculated from Eq. (2). If host i cannot synchronize automatically, a_i is set to infinity.

3.2.3. ASP operation

The operation of a host i running ASP can be formally described as follows.

Automatic Synchronization Procedure

1. In each beacon interval, host i checks whether its $c_i = p_i$. If so, host i will attempt to transmit a beacon in this beacon interval (follow the operations of IEEE 802.11). Also, the variable c_i is reset to zero. If not, go to step 4.
2. Cancel the random delay timer if a beacon is received from other mobile host before the timer has expired. The TSF timer information in the received beacon is recorded in host i's Neighbor_Table. If the timestamp in the received beacon is later than its TSF timer, host i will synchronize to this timestamp and increase its Seq_No. Simultaneously, timing information is recorded to Clock_Table. If the host that transmits the beacon is already in host i's Clock_Table, validity check (not exceeds eight beacon intervals and has the same Seq_No) will be applied. If passed, calculate a_i using Eq. 2. If mobile host i already has a value a_i, the smaller one will be selected in order to synchronize with the faster TSF timer.
3. Send a beacon out if the random delay timer has expired and no beacon has arrived during the delay period.
4. At the end of a beacon interval, increases c_i by 1.
5. Mobile host i automatically adjusts its clock *offset* 1 μs ahead in every a_i microseconds.

We use an example to illustrate how to get the interval a_i. Assume that the length of a beacon interval is 0.1 s (100,000 μs). For ease to understand, we further assume that a beacon is transmitted in the first slot of the beacon generation window and the beacon transmission time is ignored. Suppose that there are three hosts, A, B, and C in a MANET. Hosts A and B, hosts B and C are within each other's transmission range but hosts A and C can not hear from each other. Host A's, B's, and C's TSF timers start with time zero and their Seq_No's are set to zero initially. All of these hosts claim that their clock oscillate once per microsecond. But in reality, host A oscillates faster than host B which in turn oscillates faster than host C. Assume that, after one beacon interval (according to host A's clock), host A's clock oscillates 100,000 times, host B's clock oscillates 99,995 times and host C's clock oscillates 99,990 times. That is, host B's and host C's clocks are five and ten ticks slower than that of host A's every beacon interval, respectively. As shown in Figure 2, in the first beacon interval, host B successfully transmits a beacon with timestamp and Seq_No are both zero. Since the timestamp in the beacon is not later than their own, host A and C will not change their TSF timer.

Next, in the second beacon interval, host B transmits a beacon with timestamp 100,000 and Seq_No zero when its clock equals to 100,000. After receiving this beacon, host A will not modify its *offset* since the timestamp in the beacon is not faster than its own. On the contrary, host C will synchronize to this timestamp and increase its Seq_No by one. Host C achieves synchronism by changing its *offset* to five (timestamp in the received beacon − its own clock value = 100,000 − 99,995 = 5). In addition, timing information will be stored to host C's Clock_Table with MH_Id = host B, Seq_No = 0, $Last\_Recv\_Clk$ = 100,000, and $Last\_My\_Clk$ = 99,995.

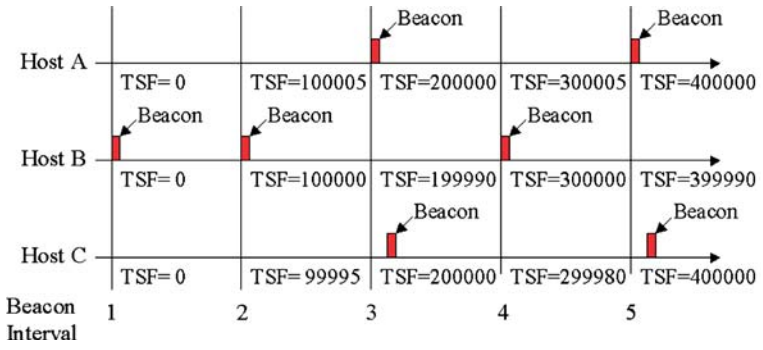

Figure 2. An example of beacon transmission.

In the third beacon interval, host A transmits a beacon with timestamp 200,000 and Seq_No zero when its clock is 200,000. Host B receives this beacon at its clock value 199,990. After receiving host A's TSF, host B increases one to its Seq_No and adds ten (200,000 − 199,990 = 10) to its *offset*. This timing information is recorded with MH_Id = host A, Seq_No = 0, $Last\_Recv\_Clk$ = 200,000, and $Last\_My\_Clk$ = 199,990. In this beacon interval, host C does not receive any beacon so it transmits a beacon when its clock is 199,995 (TSF timer is 200,000 since its *offset* is set to 5 at the second beacon interval). This beacon is received by host B successfully. But host B will not do anything because it does not contain a faster timestamp.

In the fourth beacon interval, host B transmits its beacon again with timestamp 300,000 (299,990 + 10) and Seq_No 1. Similar to the second beacon interval, host A will do nothing but host C will synchronize to this beacon. The *offset* of host C will be set to 25 (300,000 − 299,975 = 25). Also, timing information is stored to host C's Clock_Table with MH_Id = host B, Seq_No = 1, $Last\_Recv\_Clk$ = 300,000, and $Last\_My\_Clk$ = 299,975. Although host C receive two beacons from host B, host C still does not have the automatic self-time-correcting capability since these two beacons have different Seq_No values.

Lastly, in the fifth beacon interval, host A transmits its beacon which contains timestamp 400,000 and Seq_No zero, and is received by host B. Since it is a beacon with later timestamp, host B will synchronize to it by updating its *offset* to 20 (400,000 − 399,980 = 20) and increases its Seq_No by one. Again, timing information is recorded with MH_Id = host A, Seq_No = 0, $Last\_Recv\_Clk$ = 400,000, and $Last\_My\_Clk$ = 399,980. This is the second beacon from the host A with the same Seq_No hence host B can calculate its interval a_B for automatic self-time-correction:

$$Pass\_Time1 = (399{,}980 - 199{,}990) = 199{,}990,$$

$$Pass\_Time2 = (400{,}000 - 200{,}000) = 200{,}000,$$

$$Diff = Pass\_Time1 - Pass\_Time1 = 10, \text{ and}$$

$$a_{\mathrm{B}} = \left\lfloor \frac{199{,}990}{10} \right\rfloor = 19{,}999.$$

That is, host B shall automatically adjusts its *offset* by one in every 19,999 oscillations to synchronize with the host A.

4. Simulation Results

The proposed ASP is evaluated by the *ns-2* [8] simulator (CMU wireless and mobile extensions [9]). We use 224 μs as the maximum tolerable clock drift since it is the duration, specified in the IEEE 802.11 standard, for the PHY to hop to another frequency in FHSS. Asynchronism happens when a host's TSF timer is behind that of another host's over 224 μs. It is checked in every beacon interval. A beacon interval is 0.1 s long and the differences in accuracy of hosts' clocks are uniformly distributed in the range of [−0.01%, +0.01%]. The number of mobile hosts is 100, 300, or 500. Each of them is randomly located in a region of $1000 \times 1000\,\text{m}^2$ with a transmission range of 250 m. All hosts move according to the *random way-point* model [10] with maximum speed 5 m/s and pause time 50 s. We simulate the DSSS environment. Each point in the figures is the average of 10 simulation runs with simulation time 500 s (5000 beacon intervals).

In the following, we make observations from four aspects.

4.1. Effect of α

In this experiment, we investigate the effect of α to the performance of ASP by varying it from one to five. The metric is the average maximum clock drift between every pair of hosts. According to Eq. (1), α controls the number of mobile hosts to transmit beacon. A large α will reduce this number. In Figure 3, when the number of mobile hosts is 100, the clock drifts are all below 100 μs for all five different α values, among them $\alpha = 5$ performs the best. When the number of mobile hosts is 300, $\alpha = 3$ performs the best which is in turn followed by $\alpha = 1, 5, 4$, and 2, respectively. When the number of hosts is increased to 500, $\alpha = 3$ still outperform the others. Since $\alpha = 3$ performs well in most situations, $\alpha = 3$ is used in the following simulations.

4.2. Effect of Number of Hosts

In this experiment, we compare the performance of our ASP with IEEE 802.11 and ATSP on different number of hosts. The metrics used here are the maximum clock drift of any two mobile hosts and the accumulated number of asynchronisms. In Figure 4(a), where IEEE 802.11 TSF is applied over 100 hosts, the values of the maximum clock drift vary acutely. The clock drift is over 500 μs for four times and the average is 222 μs for the entire simulated 500 s.

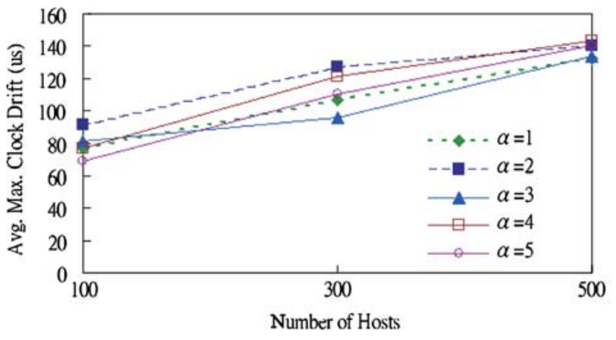

Figure 3. Effect of α.

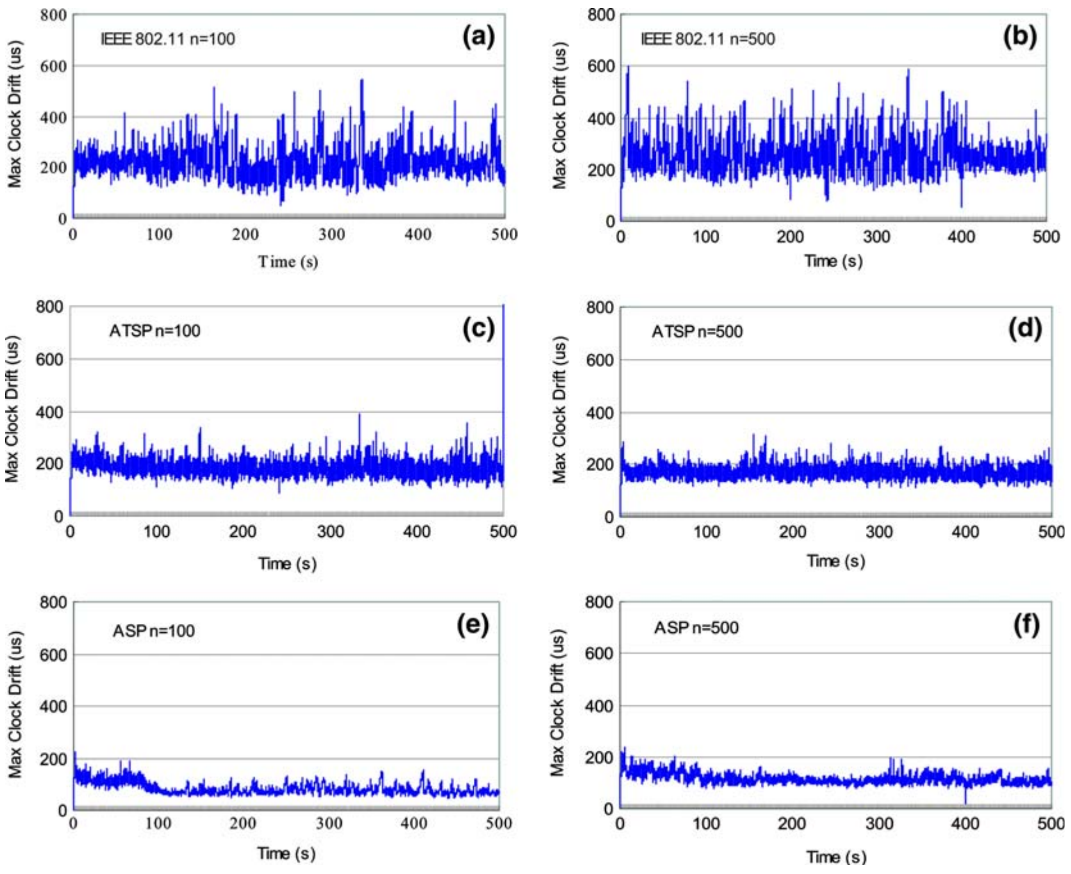

Figure 4. Maximum clock drift vs. simulation time.

Figure 4(b) shows the results when there are 500 hosts. It becomes worse with the average clock drift reaching 264 μs. The performance of the ATSP with 100 and 500 hosts are shown in Figure 4(c) and (d). Apparently it is better than IEEE 802.11 TSF; however, the values of the maximum clock drift remain high. The averages are 185 and 172 μs for 100 and 500 hosts, respectively. In contrast to IEEE 802.11 TSF and ATSP, ASP performs well as shown in Figure 4(e) and (f). In Figure 4(e), all hosts come to a stable state after simulation time 81 s. The average clock drift is 88 μs. Figure 4(f) is the result for 500 hosts. The average clock drifts is 114 μs. Our ASP reduces up to 60% and 52% over the IEEE 802.11 and ATSP, respectively.

In Figure 5, we show the accumulated number of asynchronisms for different protocols. For IEEE 802.11, in Figure 5(a), the accumulated asynchronous numbers are over 3000 for all the three cases at the end of the simulation. Also, the number of asynchronisms is proportional to the number of hosts because beacon collision probability is increased accordingly. The number of asynchronisms reduced when ATSP is adopted as shown in Figure 5(b). However, the number of asynchronisms is still proportional to the number of hosts. It is interesting that the number of asychronisms is reduced as the number of hosts increases. We consider it is because the fastest host is the only one who has the highest priority which can transmit a beacon every beacon interval; all the other hosts, with the lowest priority, can contend to transmit a beacon every $I_{\max}$ beacon intervals ($I_{\max} = 10$ in this experiment). More hosts in the network means that there are more chances to relay the fastest timing in each beacon

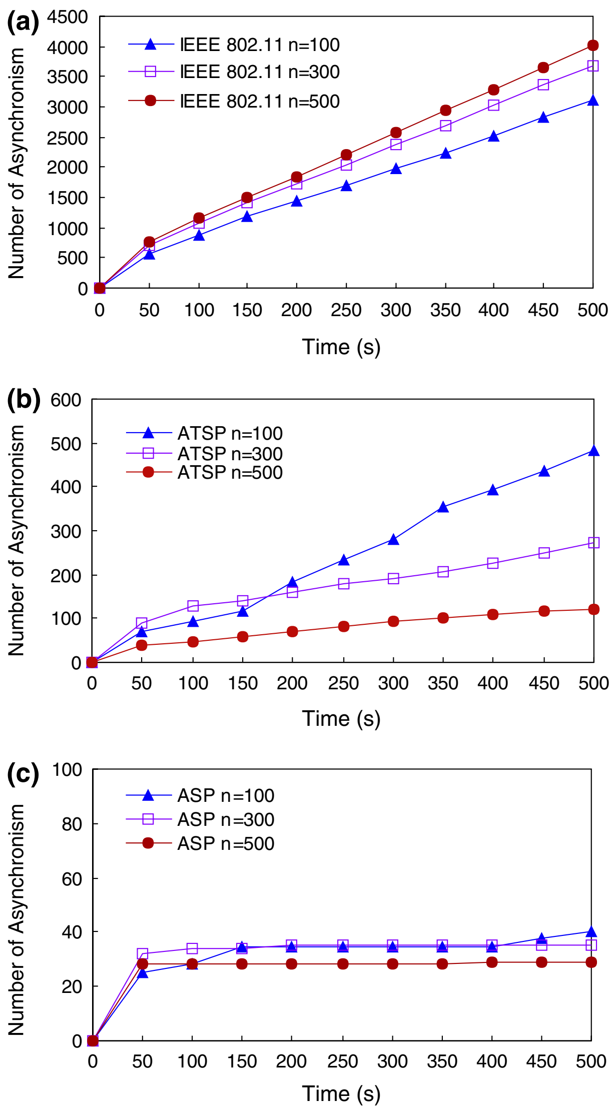

Figure 5. Accumulated number of asynchronisms vs. simulation time.

interval, thus synchronization is better achieved the number of asynchronisms is reduced. For ASP, in Figure 5(c), the whole MANET comes to a stable state after simulation time 50 s and the numbers of accumulate asynchronism are all below 40.

4.3. Effect of Mobility

Next, we investigate the effect of high mobility to the clock synchronization. The maximum speed and the pause time are set to 10 m/s and 0, respectively. In Figure 6(a) and (b), we can find that the maximum clock drifts for hosts running IEEE 802.11 still change dramatically. The average maximum clock drifts are 209 and 264 μs for 100 and 500 hosts, respectively. In Figure 6(c) and (d), hosts running ATSP achieve better synchronization. The average maximum clock drifts are 184 and 176 μs for 100 and 500 hosts, respectively. In Figure 6(e), the

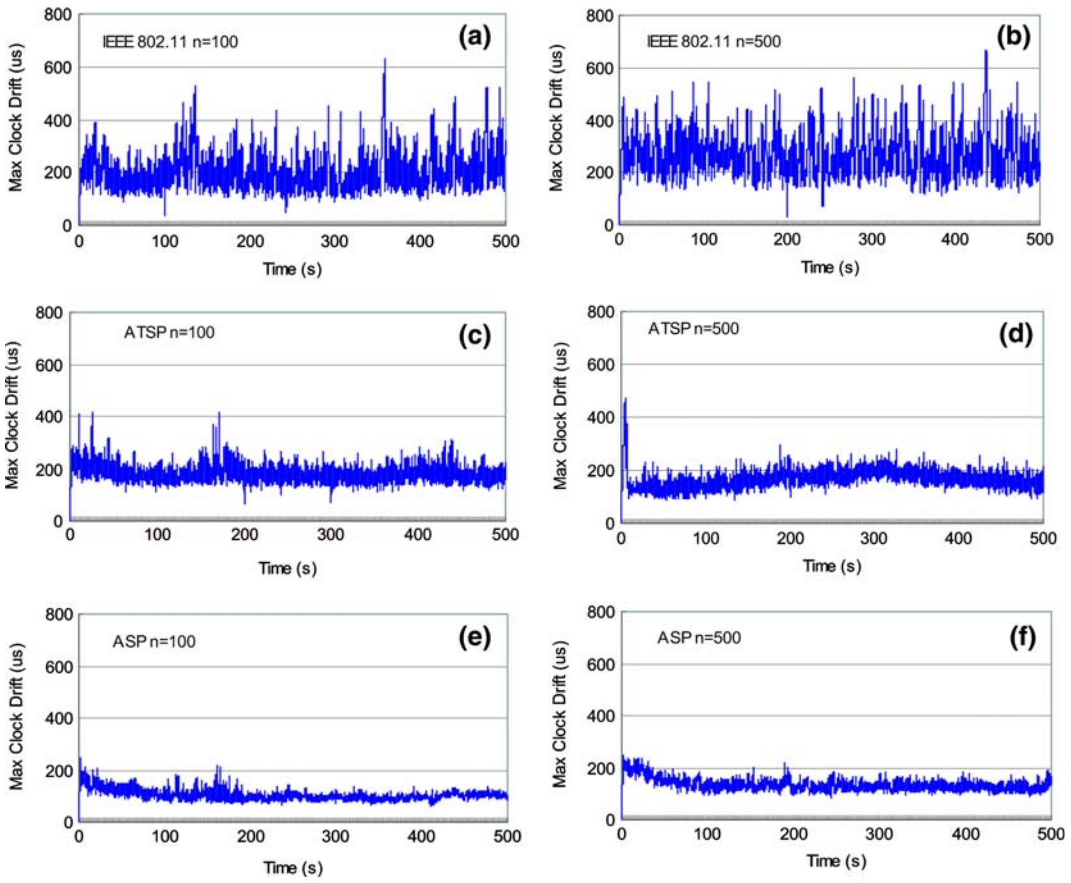

Figure 6. Maximum clock drift vs. simulation time with high mobility.

average clock drift for ASP with 100 hosts is 108 μs. A notably large clock drift (214 μs) is happened at simulation time 168 s. We believe it is because the higher mobility increases the possibility that a MANET is partitioned into several subnetworks. Such subnetworks may have large clock differences and increase the maximum clock drift when they are merged. Figure 6(f) demonstrates the result when the host number is 500. The average clock drifts for ASP with 500 hosts is 114 μs. In general, higher mobility makes the synchronization task harder because hosts would have different set of neighbors during their operation.

In Figure 7(a), we can see that asynchronism is still serious. The accumulated asynchronous numbers at simulation time 500 s are 2477, 3453, and 3523 for hosts 100, 300, and 500, respectively. For the ATSP in Figure 7(b), the accumulated asynchronous numbers at simulation time 500 s are 337, 286, and 198 for hosts 100, 300, and 500, respectively. The ASP, as shown in Figure 7(c), performs much better than the IEEE 802.11 and ATSP. When the number of hosts is under 300, the accumulated asynchronous numbers are below 40. However, when hosts are increased to 500, the number of asynchronisms upraises to 118 at the end of the simulation. It is because high mobility increases the difficulty to achieve self-time-correction to a faster host (recall that two beacons with the same Seq_No are needed from the same faster host to enable the automatic synchronization ability). Thus the improvements of ASP over the IEEE 802.11 and ATSP are somewhat reduced. Form these results, we verify that our ASP outperforms the other two protocols in handling the mobility issue.

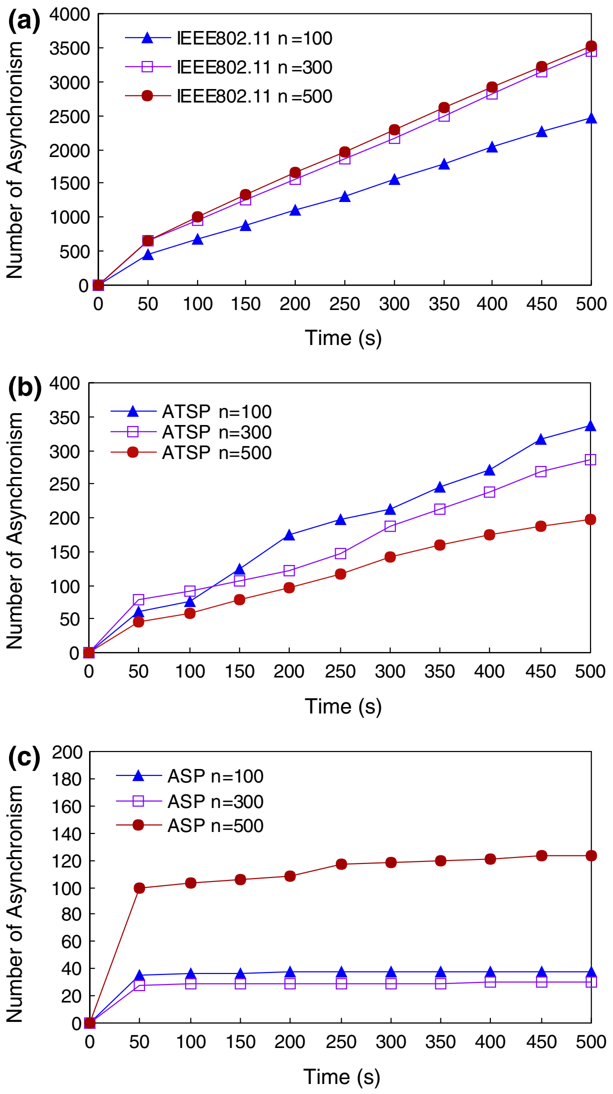

Figure 7. Accumulated number of asynchronisms vs. simulation time with high mobility.

4.4. Effect of Time-varying Clock Drift

Since hardware is not perfect, the oscillation frequency of each node's clock is time-variant. This experiment is to verify the effectiveness of our ASP in a time-varying clock drift environment. The number of hosts is 100 in this experiment. In Figure 8(a) and (b), each host's clock drift is uniformly distributed between $-50\,\mu$s/s and $50\,\mu$s/s.[1] In our simulation, clock drifts are changed every second. The average maximum clock drifts between hosts are 161 and 75 μs for ATSP and ASP, respectively. In Figure 8(c) and (d), each host's clock drift is set to $\pm 80\,\mu$s/s. Again, our ASP performs the better. The average maximum clock drifts

[1] We refer to the ECX-3S crystals (7.3728 MHz) produced by ECS Inc. International. The drift unit of the crystals is PPM.

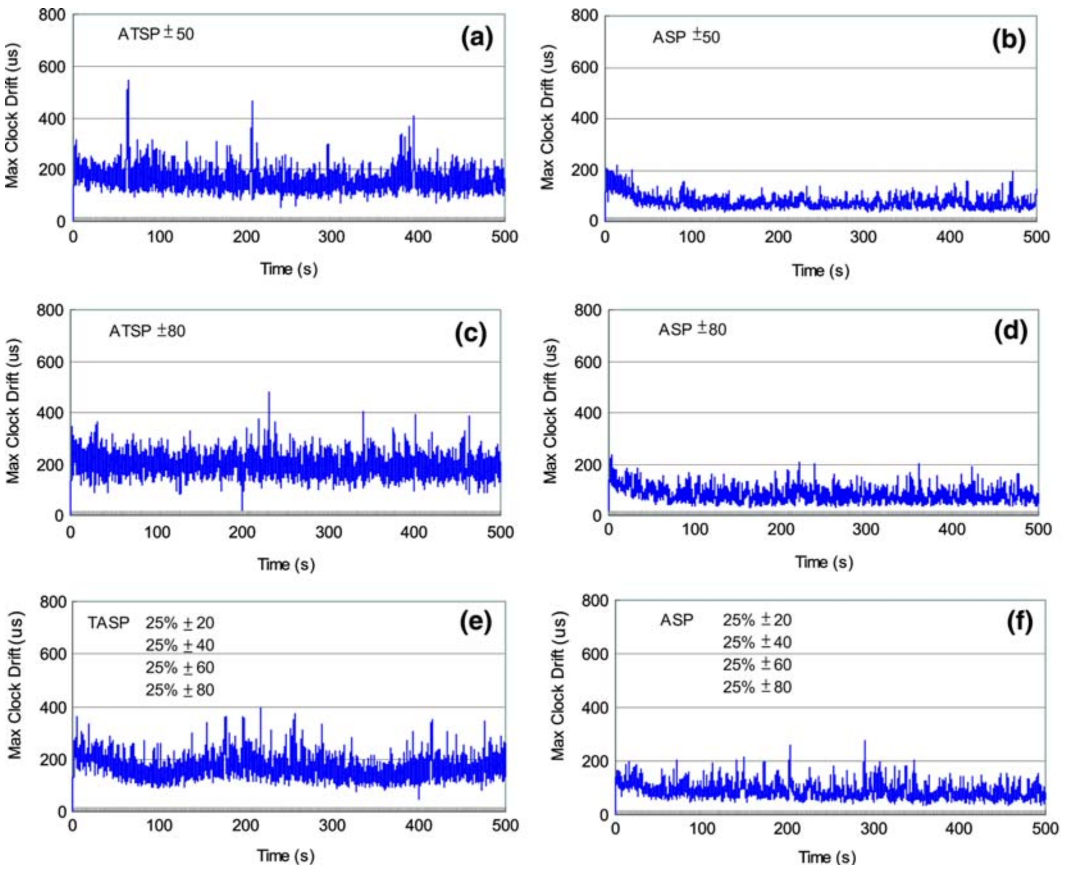

Figure 8. Maximum clock drift vs. simulation time with time-varying clock drift.

are 194 and 81 μs for ATSP and ASP, respectively. Figure 8(e) and (f) are the results where hosts' clock drifts are different to each other. We test four different clock drifts: ±20, ±40, ±60, or ±80 μs/s; one fourth of the 100 hosts are assigned for each clock drift. The average maximum clock drifts are 167 and 88 μs for ATSP and ASP, respectively. These results reveal that the performance of both ATSP and ASP are deteriorated due to time-variant clock drift. The inaccuracy makes the task of synchronization harder. Fortunately, our ASP still keep the network synchronized. It is because our ASP is built on top of the original TSF of 802.11. The original TSF still works for synchronization. The self-time-correcting function of TSP is an enhancement, which can always be corrected by newly received beacons.

5. Conclusions

A clock synchronization mechanism (ASP) in a multihop MANET is proposed. The ASP makes hosts with faster TSF timers transmit frequently. Also, after receiving the faster timing information twice from the same host with the same Seq_No, a slower host can synchronize to the faster one by increasing its TSF timer periodically. Such a mechanism reduces the necessity of the beacons from the faster-clock hosts. Furthermore, a host who has the ability to synchronize automatically can take on the job to spread the fast timing out, which synchronizes the whole MANET soon. Simulation results show that our ASP achieves great improvement over the IEEE 802.11 TSF and ATSP. Even in a very large MANET with highly mobile hosts, the ASP keeps the system in a well synchronized manner.

References

1. Chih-Min Chao, Jang-Ping Sheu, and I-Cheng Chou, "An Adaptive Quorum-Based Energy Conserving Protocol for IEEE 802.11 Ad Hoc Networks", *IEEE Trans. on Mobile Computing*, Vol. 5, No. 5, pp. 560–570, May 2006.
2. IEEE 802.11 Standard (IEEE Computer Society LAN MAN Standards Committee), *Wireless LAN Medium Access Control and Physical Layer Specifications*, Aug. 1999.
3. G. Bianchi, L. Fratta, and M. Oliveri, "Performance Evaluation and Enhancement of the CSMA/CA MAC Protocol for 802.11 Wireless LANs", in *Proceedings of the 7th International Symposium on Personal, Indoor and Mobile Radio Communications*, Tainan, Taiwan, pp. 392–396, Oct. 1996.
4. J. Li, C. Blake, D.S.J.D. Couto, H.I. Lee, and R. Morris, "Capacity of Ad Hoc Wireless Networks", in *Proceedings of the 7th ACM International Conference on Mobile Computing and Networking*, Rome, Italy, pp. 61–69, Jul. 2001.
5. Y.C. Tay and K.C. Chua, "A Capacity Analysis for the IEEE 802.11 MAC Protocol", *ACM Wireless Networks*, Vol. 7, No. 2, pp. 159–171, Apr. 2001.
6. L. Huang and T.H. Lai, "On the Scalability of IEEE 802.11 Ad Hoc Networks", in *Proceedings of the third ACM International Symposium on Mobile Ad Hoc Networking & Computing*, Lausanne, Switzerland, pp. 173–182, Jun. 2002.
7. T.H. Lai and D. Zhou, "Efficient and scalable IEEE 802.11 Ad-Hoc-Mode Timing Synchronization Function", in *Proceedings of the 17th International Conference on Advanced Information Networking and Applications*, pp. 318–323, Mar. 2003.
8. *The Network Simulator – ns-2*, http://www.isi.edu/nsnam/ns/, 2006.
9. *The CMU Monarch Project's Wireless and Mobility Extensions to ns*, http://www.monarch.cs.cmu.edu/, Aug. 1998.
10. J. Broch, D. Maltz, D. Johnson, Y. Hu, and J. Jetcheva, "A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols", in *Proceedings of the 4th Annual ACM/IEEE International Conference on Mobile Computing and Networking*, Dallas, TX, USA, pp. 85–97, Oct. 1998.

Jang-Ping Sheu received the B.S. degree in computer science from Tamkang University, Taiwan, Republic of China, in 1981, and the M.S. and Ph.D. degrees in computer science from National Tsing Hua University, Taiwan, Republic of China, in 1983 and 1987, respectively.

He joined the faculty of the Department of Electrical Engineering, National Central University, Taiwan, Republic of China, as an Associate Professor in 1987. He is currently a Professor of the Department of Computer Science and Information Engineering and Director of Computer Center, National Central University. He was a Chair of Department of Computer Science and Information Engineering, National Central University from 1997 to 1999. He was a visiting professor at the Department of Electrical and Computer Engineering, University of

California, Irvine from July 1999 to April 2000. His current research interests include wireless communications, mobile computing and parallel processing. He was an associate editor of Journal of the Chinese Institute of Electrical Engineering, from 1996 to 2000. He was an associate editor of Journal of Information Science and Engineering from 1996 to 2002. He was an associate editor of Journal of the Chinese Institute of Engineers from 1998 to 2004. He is an associate editor of the IEEE Transactions on Parallel and Distributed Systems and International Journal of Ad Hoc and Ubiquitous Computing. He has served as a Program Chair and Vice Program Chair for a number of international conferences including IEEE ICPADS'02, ICPP'03, and IEEE MSN'05.

He received the Distinguished Research Awards of the National Science Council of the Republic of China in 1993–1994, 1995–1996, and 1997–1998. He was the Specially Granted Researchers, National Science Council, from 1999 to 2005. He received the Distinguished Engineering Professor Award of the Chinese Institute of Engineers in 2003. He received the Distinguished Professor award of the National Central University in 2005. Dr. Sheu is a senior member of the IEEE, a member of the ACM, and Phi Tau Phi Society.

Chih-Min Chao received the B.S. and M.S. degrees in computer science from Fu-Jen Catholic University and National Tsing-Hua University in 1992 and 1996, respectively, and the Ph.D. degree in computer science and information engineering from National Central University in January of 2004. He was with SENAO International in 1996. He was an assistant professor at the TamKang University, Taiwan, from 2004 to 2005. Since 2005, he has been an assistant professor with the Department of Computer Science and Engineering, National Taiwan Ocean University, Taiwan. His research interests include mobile computing and wireless communication.

Wei-Kai Hu received the B.S. degree in Computer Science and Information Engineering from Tunghai University, Taichung, Taiwan, in 2001. He now studies for master and Ph.D. degree in Department of Computer Science and Information Engineering of National Central University, meanwhile. His current research interests include wireless sensor networks, Ad Hoc networks.

Ching-Wen Sun received the B.S. degree in computer and information science from Soochow University in 2001 and the M.S. degree in computer science and information engineering from the National Central University in 2003, respectively. She worked in AsusTek Computer Inc. from 2003 until now. Her research domains include communications of wireless LAN and sensor network. She currently focuses on the communicaitons of GSM/GPRS/UMTS.