will be faster in our paradigm, given the summaries and the generalized attributes. Furthermore, we have shown that the number of involved brokers during event processing is, on average, smaller.

6. Conclusions

We contributed a new paradigm for pub/sub systems based on the novel notion of subscription summaries. We have presented the data structures constituting the summaries and the algorithms manipulating these summaries in order to match incoming events against the brokers' subscriptions. Also, we have developed the notion of multi-broker summaries, shown how to compute them, and the accompanying algorithm for subscription summary propagation. These contributions introduce significant performance gains during subscription propagation, for network bandwidth, storage requirements, and in required broker involvement (hop counts). We then contributed a distributed event processing algorithm, which ensures efficiency during the event processing phase, in terms of the number of brokers involved in event routing.

Our performance results show that our approach can drastically improve the bandwidth requirements to propagate subscriptions, outperforming the subsumption mechanism of Siena by a factor of four to eight. At the same time, the hop count for both subscription propagation and event processing is smaller and the computational requirements at each broker for filtering and matching events are expected to be better than those of related work. Finally, the storage requirements are smaller than Siena's by 2 to 5 times.

Our on-going work includes ensuring load balancing during event processing, which is an open problem. We employ *'virtual degrees'* for the maximum-degree nodes, reducing their load, while continuing, however, to offer significant improvements. Further, we are currently extending our structures to accommodate dynamically-changing attribute schemata, for larger-scale networks (e.g., multi-ISP, global CDNs) (basically, this only requires changing the c_3 field of subscription ids). Finally, we are also currently developing techniques combining summarization and subsumption.

7. References

[1] M. K. Aguilera, R. E. Strom, D. C. Sturman, M. Astley, and T. D. Chandra. Matching events in a content-based subscription system. Proc. ACM PODC Symposium, pp 53–61, 1999.

[2] G. Banavar, T. Chandra, B. Mukherjee, J. Nagarajarao, R. E. Strom, and D. C. Sturman. An efficient multicast protocol for content-based publish/subscribe systems. *Proc. ICDCS*, pp 262–272, 1999.

[3] G. Banavar, M. Kaplan, K. Shaw, R. E. Strom, D. C. Sturman, and W. Tao. Information flow based event distribution middleware. *Proc. ICDCS Workshop on Electronic Commerce and Web Applications*, 1999.

[4] Cable and Wireless plc. http://www.cw.com.

[5] A. Carzaniga, E. Nitto, D. Rosenblum, and A. Wolf. Issues in supporting event-based architectural styles. *3rd Intl Software Architecture Workshop*, 1998.

[6] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Achieving scalability and expressiveness in an Internet-scale event notification service. *Proc. ACM PODC*, pp 219–227, 2000.

[7] G. Cugola, E. D. Nitto, and A. Fuggetta. The JEDI event-based infrastructure and its application to the development of OPSS WFMS. *IEEE TSE*, Sep. 2001.

[8] F. Fabret, H. A. Jacobsen, F. Llirbat, J. Pereira, K. A. Ross, and D. Shasha. Filtering algorithms and implementation for very fast publish/subscribe systems. *ACM SIGMOD 2001*, pp. 115-126, 2001.

[9] J. Gough and G. Smith. Efficient recognition of events in a distributed system. *Proc. Australasian Computer Science Conference*, Feb. 1995.

[10] R. E. Gruber, B. Krishnamurthy, and E. Panagos. The Architecture of the READY Event Notification Service. *ICDCS workshop*, June 1999.

[11] G. Muhl, L. Fiege, F. Gartner, and A. Buchmann. Evaluating advanced routing algorithms for content-based publish/subscribe systems. *Proc. MASCOTS'02*, pp167-176, 2002.

[12] Object Management Group. CORBAservices – event service specification. Technical report 2001. ftp://ftp.omg.org/pub/docs/formal/01-03-01.pdf.

[13] Object Management Group. CORBAservices – notification service specification. Tech. report, 2000. ftp://ftp.omg.org/pub/docs/formal/00-06-20.pdf.

[14] B. Oki, M. Pfluegl, A. Siegel, and D. Skeen. The Information Bus - an architecture for extensible distributed systems. Operating Systems Review, 27.5:58–68, 1993.

[15] P. R. Pietzuch and J. Bacon. A distributed event-based middleware architecture. *Proc. 1st International Workshop on Distributed Event-Based Systems*, 2002.

[16] A. Rowstron and P. Druschel. Pastry: Scalable decentralized object location and routing for large-scale peer-to-peer systems. *Proc. Middleware 01*, 2001.

[17] B. Segall and D. Arnold. Elvin has left the building: A publish/subscribe notification service with quenching. *Proc. Australian UNIX Users Group Technical Conference*, pp 243–255, 1997.

[18] SoftWired Inc. iBus. http://www.softwired-inc.com.

[19] Sun Microsystems, Inc. Jini(TM) technology core platform spec - distributed events. Technical report, 2000. http://www.sun.com/jini/specs.

[20] TIBCO Inc. TIB/Rendezvous. http://www.tibco.com.

[21] P. Triantafillou, A. Economides. Subscription Summaries for Scalability and Efficiency in Publish/Subscribe Systems. *Proc. 1st International Workshop on Distributed Event-Based Systems*, 2002.

[22] Vitria. BusinessWare. http://www.vitria.com

A Clock Synchronization Algorithm for Multi-Hop Wireless Ad Hoc Networks *

Jang-Ping Sheu, Chih-Min Chao, and Ching-Wen Sun
Department of Computer Science and Information Engineering
National Central University, Taiwan
Email: sheujp@csie.ncu.edu.tw, cmchao@axp1.csie.ncu.edu.tw, cwsun@axp1.csie.ncu.edu.tw

Abstract

In multi-hop 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 multi-hop environment. Our ASP has two features. Firstly, a faster host has higher priority to send its timing information out than a slower one. Secondly, 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 multi-hop wireless ad hoc networks.

Keywords: Clock synchronization, IEEE 802.11, multi-hop 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. A distributed *timing synchronization function (TSF)* is specified in IEEE 802.11 WLAN standard [1] to fulfill clock synchronization in a MANET. In this synchronization mechanism, each mobile host is responsible for exchanging timing information through the periodically 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 [2, 3, 4]). A scalability problem is also induced by this IEEE 802.11 standard TSF due to the beacon contentions [5, 6]. 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 [5] 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 synchronize others. Thus, ATSP gives the fastest host (the host that has the fastest timing) the highest priority to transmit beacons (by increasing its beacon transmission frequency). On the contrary, slower hosts' beacon transmission frequencies are reduced. 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 microseconds. To overcome these problems, a revision of ATSP, which is called *Tiered Adaptive Timing Synchronization Procedure (TATSP)*, is proposed in [6]. Both ATSP and TATSP achieve clock synchronization for mobile hosts and scale well in a single-hop MANET. However, these two algorithms fail to apply to a multi-hop MANET.

In this paper, we propose a clock synchronization algorithm, which is called *automatic self-time-correcting procedure (ASP)*, to achieve clock synchro-

*This work was supported by the Ministry of Education, the Republic of China, under Grant A-92-H-FA07-1-4 (Learning Technology).

nization among mobile hosts and hence to solve the scalability problem in a multi-hop MANET. Two tasks must be done to fulfill clock synchronization in a multi-hop 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 [1] 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 × 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.

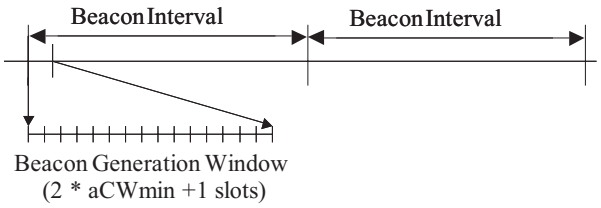

Figure 1. Beacon generation window.

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

3. Clock Synchronization in Multi-Hop MANETs

As mentioned earlier, to achieve clock synchronization in multi-hop 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 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'

Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04)
1063-6927/04 $20.00 © 2004 **IEEE**

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.

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

where 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)

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

$$a_i = \lfloor \text{Pass\_time1} \ / \ \text{Diff} \rfloor \qquad (2)$$

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 multi-hop 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.

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

Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04)
1063-6927/04 $20.00 © 2004 **IEEE**

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 can not synchronize automatically, a_i is set to infinity.

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 (exceeds eight beacon intervals or not) and a_i calculation will be applied. If mobile host i already has a value a_i, the small one will be selected.

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* one microsecond ahead in every a_i microseconds.

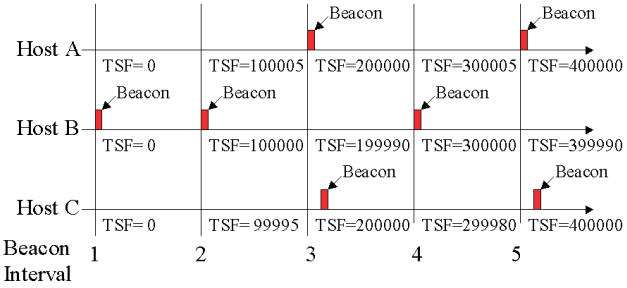

Figure 2. An example of beacon transmission

We use an example to illustrate how to get the interval a_i. Assume that the length of a beacon interval is 0.1 second (100000 microseconds). 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 100000 times, host B's clock oscillates 99995 times and host C's clock oscillates 99990 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 Fig. 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

Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04)
1063-6927/04 $20.00 © 2004 **IEEE**

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 100000 and Seq_No zero when its clock equals to 100000. 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 = 100000 - 99995 = 5). In addition, timing information will be stored to host C's Clock_Table with MH_Id = host B, Seq_No = 0, $Last\_Recv\_Clk$ = 100000, and $Last\_My\_Clk$ = 99995.

In the third beacon interval, host A transmits a beacon with timestamp 200000 and Seq_No zero when its clock is 200000. Host B receives this beacon at its clock value 199990. After receiving host A's TSF, host B increases one to its Seq_No and adds ten (200000 - 199990 = 10) to its *offset*. This timing information is recorded with MH_Id = host A, Seq_No = 0, $Last\_Recv\_Clk$ = 200000, and $Last\_My\_Clk$ = 199990. In this beacon interval, host C does not receive any beacon so it transmits a beacon when its clock is 199995 (TSF timer is 200000 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 300000 (299990 + 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 (300000 - 299975 = 25). Also, timing information is stored to host C's Clock_Table with MH_Id = host B, Seq_No = 1, $Last\_Recv\_Clk$ = 300000, and $Last\_My\_Clk$ = 299975. 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 400000 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 (400000 - 399980 = 20) and increases its Seq_No by one. Again, timing information is recorded with MH_Id = host A, Seq_No = 0, $Last\_Recv\_Clk$ = 400000, and $Last\_My\_Clk$ = 399980. 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:

$$
\begin{aligned}
Pass\_Time1 &= (399980 - 199990) = 199990, \\
Pass\_Time2 &= (400000 - 200000) = 200000, \\
Diff &= Pass\_Time1 - Pass\_Time1 = 10, \text{ and} \\
a_B &= \left\lfloor \frac{199990}{10} \right\rfloor = 19999.
\end{aligned}
$$

That is, host B shall automatically adjusts its *offset* by one in every 19999 oscillations to synchronize with the host A.

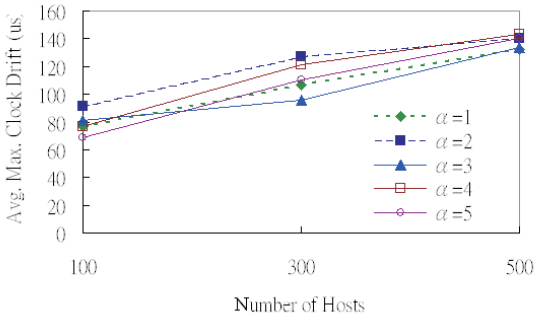

Figure 3. Effect of α

4. Simulation Results

The proposed ASP is evaluated by the *ns-2* [7] simulator (CMU wireless and mobile extensions [8]). 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 second long and the clock accuracy of hosts is 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 × 1000 square meters with a transmission range of 250 meters. All hosts move according to the *random way-point* model [9] with maximum speed 5 m/s and pause time 50 seconds. We simulate the DSSS environment. Each point in the Figs. is the average of ten simulation runs with simulation time 500 seconds (5000 beacon intervals).

In the following, we make observations from three aspects.

(A) Effect of α:

In this experiment, we investigate the effect of α to the performance of ASP by varying it from one

Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04)
1063-6927/04 $20.00 © 2004 **IEEE**

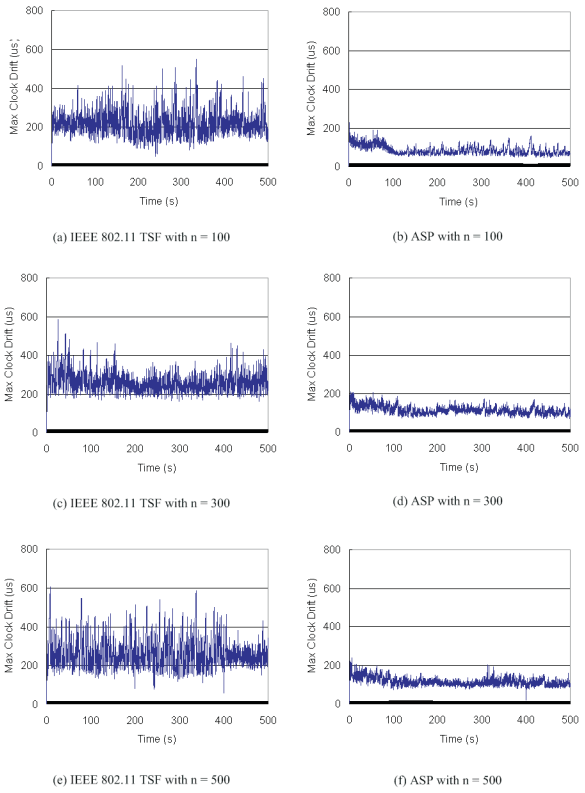

Figure 4. Maximum clock drift vs. simulation time

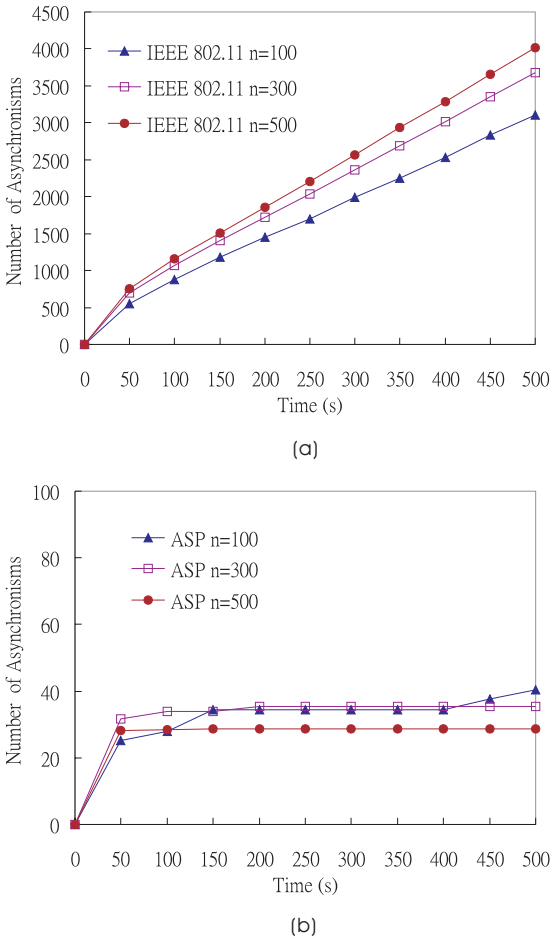

Figure 5. Accumulated number of asynchronisms vs. simulation time

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 Fig. 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.

(B) Effect of Number of Hosts: In this experiment, we compare the performance of our ASP with IEEE 802.11 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 Fig. 4 (a), where IEEE 802.11 TSF is applied over totally 100 hosts, the maximum clock drift varies acutely. The clock drift is over 500 μs for four times and the average clock drift is 222 μs for the entire simulated 500 seconds. In contrast to IEEE 802.11 TSF, ASP performs well as shown in Fig. 4 (b). All hosts come to a stable state after simulation time 81 seconds. The average clock drift is 88 μs. Fig. 4 (c) and (d), reveal similar results with 300 hosts. The average clock drifts for IEEE 802.11 and ASP are 260 μs and 97 μs, respectively. Fig. 4 (e) and (f) show the results when host number is 500. Again, ASP performs much better than IEEE 802.11. The average clock drifts for IEEE 802.11 and ASP are 264 μs and 114 μs, respectively. Our ASP reduces about 60% of the average clock drift.

In Fig. 5, we see the accumulated number of asynchronisms for IEEE 802.11 is far larger than that of ASP. For IEEE 802.11, in Fig. 5 (a), the ac-

Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04)
1063-6927/04 $20.00 © 2004 **IEEE**

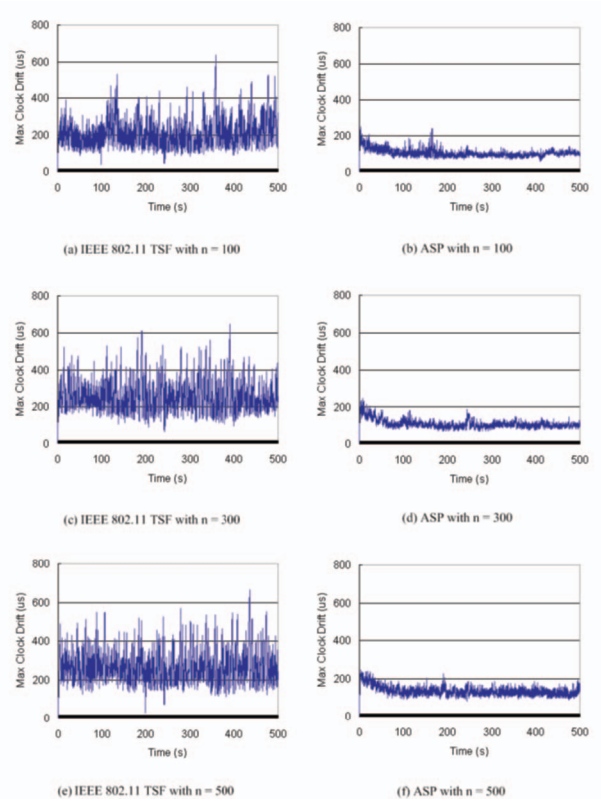

Figure 6. Maximum clock drift vs. simulation time with high mobility

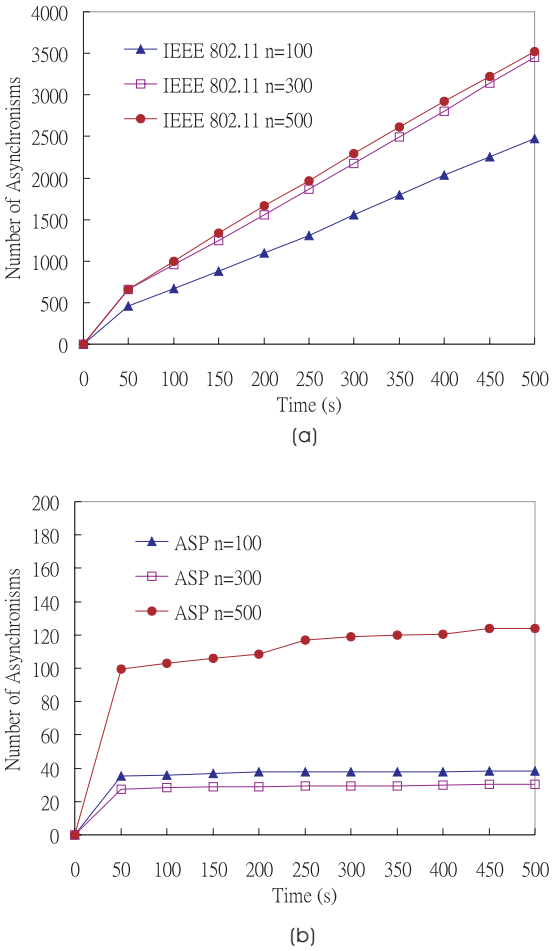

Figure 7. Accumulated number of asynchronisms vs. simulation time with high mobility

cumulated asynchronous numbers are over 3000 for all the three cases at the end of the simulation. Also, the number of asynchronism is proportional to the number of hosts because beacon collision probability is increased accordingly. For ASP, in Fig. 5 (b), the whole MANET comes to a stable state after simulation time 50 seconds and the numbers of accumulate asynchronism are all below 40. From these experiments, we conclude that clock synchronization problem is severer with larger hosts in IEEE 802.11 while our ASP works well to keep the hosts synchronized even in a very large MANET.

(C) Effect of Mobility: Lastly, 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 Fig. 6 (a), we can find that the maximum clock drift for 100 hosts running IEEE 802.11 still changes dramatically with the highest value 632 μs at simulation time 358 seconds. The average maximum clock drift is 209 μs. In Fig. 6 (b), the average clock drift for ASP is 108 μs. A notably large clock drift (214 μs) is happened at simulation time 168 seconds. We believe it, because the higher mobility increases the possibility that a MANET can be partitioned into several subnetworks. Such subnetworks may have large clock differences and increase the maximum clock drift when they are merged. Fig. 6 (c) and (d), with 300 hosts, have similar results. The average clock drifts for IEEE 802.11 and ASP are 262 μs and 107 μs, respectively. Fig. 6 (e) and (f) demonstrate the results when host number is 500. Again, ASP performs much better than the IEEE 802.11. The average clock drifts for IEEE 802.11 and ASP are 264 μs and 114 μs, respectively.

In Fig. 7 (a), we can see that asynchronism is still serious. The accumulated asynchronous numbers

Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04)
1063-6927/04 $20.00 © 2004 **IEEE**