

Accurate Clock Synchronization for IEEE 802.11-Based Multi-Hop Wireless Networks

Jui-Hao Chiang Tzi-cker Chiueh
Computer Science Department
Stony Brook University
Email: {j-chiang, chiueh}@cs.sunysb.edu

Abstract—Clock synchronization is an essential building block for many control mechanisms used in wireless networks, including frequency hopping, power management, and packet scheduling. Although the IEEE 802.11 standard [1] provides a clock synchronization mechanism for multi-hop wireless networks, it has well-documented accuracy problems [2], [3] because the beacon frames transmitted by nodes with faster clock are not prioritized at the MAC layer. This paper describes the design and implementation of the first known software-only solution that successfully eliminates the deficiency of IEEE 802.11's clock synchronization mechanism when operating on multi-hop wireless networks without requiring any hardware/protocol modifications. The key idea of this solution is to periodically arrange for a node to become the fastest node in the network by jumping its clock with a substantial amount, and force other nodes to synchronize their clocks with this leading node by virtue of IEEE 802.11's native clock synchronization mechanism. Empirical measurements of a working implementation of the proposed algorithm on a multi-hop wireless network tested conclusively demonstrates its effectiveness in removing IEEE 802.11's clock synchronization accuracy problem.

I. INTRODUCTION

Clock synchronization is a critical building block for distributed systems in general and wireless networked systems in particular. For example, at the protocol level, wireless networks rely on clock synchronization to support frequency hopping in spread spectrum communications, power saving-driven wireless node sleep and wake-up, and interference-reducing radio channel scheduling; at the application level, wireless sensor network applications rely on clock synchronization to form a consistent ordering among a distributed collection of sensor values and events. In addition to minimizing the clock drift among nodes, it is essential that a clock synchronization scheme incur minimum radio channel time usage because the radio resource in wireless networks is typically limited and needs to be used with maximum efficiency.

The focus of this paper is on clock synchronization algorithms for wireless networks built with the IEEE 802.11 interfaces [1], which uses special *beacon frames* to synchronize the Time Synchronization Function (TSF) counter on the wireless interfaces of the participating nodes in a single-hop or multi-hop network. There are two problems with IEEE 802.11's clock synchronization mechanism. First, this mechanism was originally designed to support the needs of IEEE 802.11 itself, particularly frequency hopping and power saving, and was never meant to be used by higher-layer network control

mechanisms or applications/services. Second, this mechanism has well-known accuracy problems [2], [3] that lead to gradual loss of synchronization among participating nodes, especially when it is used in large multi-hop wireless networks.

Because IEEE 802.11's clock synchronization mechanism is implemented directly in hardware, it is more accurate than software-based network clock synchronization mechanisms such as NTP [4], especially on single-hop wireless networks. Guo and Chiueh [5], [6] showed how to turn IEEE 802.11's clock synchronization mechanism into a general primitive that other network control mechanisms and services can be built on. In particular, they implemented a software-based TDMA (time-division multiple access) medium access control (MAC) scheme using IEEE 802.11's clock synchronization, and demonstrate its effectiveness by doubling the number of VoIP connections that can be supported within a single IEEE 802.11 channel when compared with IEEE 802.11 and IEEE 802.11e, which is designed specifically for supporting multimedia traffic.

Although IEEE 802.11's clock synchronization mechanism is both efficient and effective in the *infrastructure* mode, its effectiveness in the *multi-hop ad hoc* mode is less than satisfactory, because an IEEE 802.11-based multi-hop network occasionally may be partitioned into multiple timing islands, each running a different clock [7]. The root cause of this loss of clock synchronization is that nodes with faster clocks cannot always successfully send out beacon frames that carry clock synchronization information. There are two reasons why these beacon frames fail to be sent out. First, the probability that the node with the fastest clock in a collision domain always wins in the channel access competition and thus successfully sends out its beacon frames decreases when the number of nodes in the domain increases. This is because every node in a collision domain competes on an equal footing. Second, beacon frame transmission on nodes with faster clock may be suppressed by the beacon frames sent by nodes with slower clock if the latter send out their beacon frames earlier.

Intuitively, an ideal clock synchronization scheme for a multi-hop wireless network should start the propagation of the synchronization beacon frames from the network node with the fastest clock, and force each of its neighbors to *immediately* relay the beacon frame forward whenever it receives one. However, this ideal scheme requires an expensive step of locating the node with the fastest clock, and thus does not scale

well in large multi-hop wireless networks. To approximate this ideal scheme while eliminating its limitation, we propose a *clock-jumping* approach to the multi-hop clock synchronization problem that *forces* an arbitrarily chosen wireless network node to periodically jump its clock and become the fastest node. All the other nodes simply forward a beacon with a jumped timestamp whenever they receive one. With clock-jumping, there is no need to search for the fastest network node, and both the implementation complexity and the radio channel time usage required for synchronization are reduced significantly.

The rest of this paper is organized as follows. Section 2 reviews previous research on the problem of clock synchronization for multi-hop wireless networks. Section 3 details the design and implementation of the proposed clock-jumping synchronization mechanism, while Section 4 presents the evaluation of the proposed clock-jumping synchronization algorithm that effectively rectifies the synchronization accuracy problem of the IEEE 802.11 standard. Section 5 concludes this paper with a summary of the main research contributions.

II. RELATED WORK

The IEEE 802.11 standard [1] uses a TSF counter to synchronize the nodes in a wireless network operating in the *ad hoc* mode. Within an Independent Basic Service Set (IBSS), each node maintains a 64-bit TSF counter with micro-second resolution. During each beacon period, each node tries to send out a beacon frame time-stamped with its own TSF counter value *at the instant of transmission*. When a node receives a beacon from some other node, it updates its local TSF counter with the beacon's timestamp if the latter is larger, and suppresses its own beacon transmission in the current beacon period if there is one pending.

Huang and Lai [2] identified the *fastest node asynchronism* problem in the IEEE 802.11's native clock synchronization mechanism. As the number of nodes in a network increases, the node with the fastest clock has a lower chance to successfully send out its beacon frames. As a result, the fastest node's clock may keep drifting away from others' clocks, because it ignores beacons with smaller timestamps and its own beacon frame cannot reach others. To solve this problem, Huang and Lai proposed an Adaptive Timing Synchronization Procedure (ATSP) that adjusts the beacon transmission frequency of each node. In each beacon period, if a node receives a beacon with a larger timestamp, it reduces its beacon transmission frequency. Otherwise, it increases the beacon transmission frequency until reaching the maximum allowed value. With this approach, the fastest node has a higher chance to transmit beacons successfully. In contrast, the proposed clock-jumping algorithm ensures the fastest node always has a higher priority for beacon transmission, because the beacon transmission of slower nodes is triggered by faster nodes.

So and Vaidya [7] identified another problem with IEEE 802.11's native clock synchronization mechanism: *time partitioning*, in which a multi-hop network is partitioned into multiple disjoint clock islands, and these islands are out of

synchronization with one another. They proposed a Multi-hop Timing Synchronization Function (MTSF) scheme, which requires each node to maintain a path to the fastest node, and forward the beacon from the fastest node to the entire network. The key idea here is to ensure that the beacon sent by the fastest node could keep moving forward without being suppressed in the middle of the network. Each node schedules its beacon transmission every two beacon intervals, and staggers with the fastest node among its neighbors. This way, the beacon from faster nodes can be propagated farther than IEEE 802.11's scheme. Unlike the MTSF algorithm, the proposed clock-jumping algorithm does not require the expensive step of searching for the fastest node, because it has the flexibility to choose any node as the fastest node.

The Automatic Self-time-correcting Procedure (ASP) proposed by Sheu et al. [8] adjusts the beacon transmission frequency based on the relative clock values among neighbors. Faster nodes are given a higher priority to transmit beacons than slower nodes. Another feature in ASP is to require each node to estimate the clock rate difference between itself and its faster neighbor, such that its clock can be properly adjusted even when it does not receive any beacons with a larger timestamp. However, to measure the clock rate of another node is difficult in general because of frequent change in channel condition and signal propagation delay. Our algorithm does not require such measurements.

In the paper of Li and Rus [9], three methods have been proposed to synchronize clocks on wireless sensor networks. The all node-based and cluster-based synchronization methods incur too high an overhead to be usable on large multi-hop networks. In addition, the all node-based method assumes that the clock cycle on each node be the same, which is impractical. As for the diffusion-based method, it tries to decrease the number of nodes participating in local clock synchronization. Each node periodically collects clock information from its neighbors and spreads their average value back to them.

Another approach, called Reference Broadcast Synchronization (RBS) [10], requires each node to send reference beacons to its neighbors via physical layer broadcast, which does not carry explicit timestamps. Each receiver compares these broadcasted references and applies a statistics method to synchronize with each other. This method seems to achieve decent synchronization accuracy, but at the expense of substantial protocol overhead.

For wired networks, the Network Time Protocol [4] proposed by Mills has been widely used in several applications, and already been an IETF standard. In NTP, a tree hierarchy is built with several sources of clock references at the top. Between branches of the tree, packets with timestamp are exchanged for synchronization. Ganeriwal et al. [11] proposed the Timing-Sync Protocol for Sensor Networks (TPSN) to synchronize sensor network nodes. TPSN, similar to NTP, tries to reduce the clock estimation error by exchanging messages between each pair of neighboring nodes. The resulting accuracy is high but the packet exchange overhead is also substantial especially for networks with a large number of

nodes. The proposed clock-jumping algorithm does not need to explicitly build a tree structure among network nodes, and minimize the number of packets required during a synchronization transaction.

A recent work by Veitch et al. [12] tries to improve NTP with the Time Stamp Counter (TSC) register, which counts the number of clock cycles in the CPU. This paper introduces a new software clock, TSC-NTP, which makes use of NTP for synchronization. The result shows better accuracy than the original NTP because of TSC's better stability. This work originally required sophisticated kernel modifications, but their later work [13] shows that it can be implemented as a user-level software clock without modifying the kernel. Both TSC-NTP and NTP perform well on wired networks because of the use of a hierarchical structure. But they are not suitable for wireless networks because their dynamic change in physical connectivity, asymmetric link capacity/delay, and frequent link failures.

Compared with other proposals surveyed in this section, the proposed clock-jumping algorithm is simpler, more efficient, and could be readily implemented on IEEE 802.11-based multi-hop wireless networks without any hardware/protocol modifications, as demonstrated by later sections of this paper.

III. CLOCK-JUMPING APPROACH TO MULTI-HOP CLOCK SYNCHRONIZATION

When an IEEE 802.11 network operates in the *ad hoc* mode, every network node attempts to send out beacon frames, and runs the following distributed clock synchronization algorithm:

- 1) Set a beacon timer at the beginning of every beacon period, and when the timer expires, broadcast a beacon frame that contains a timestamp which is set to the local TSF counter value at the time of transmission.
- 2) Upon receiving a beacon frame, suppress its own beacon timer, and copy the frame's timestamp to the local TSF counter if the frame's timestamp is larger than the TSF counter's current value.

In theory, a multi-hop network should converge to the TSF counter value of the node with the fastest clock. In practice, this is often not the case because the fastest node does not always have the chance to send out its beacon frame in every beacon interval. Worse yet, because no priority is given to nodes with a faster clock, the probability that the fastest node is able to propagate its beacon frames to every other node in the network *decreases* when the size of the network grows. As a result, a multi-hop network may be partitioned into multiple disjoint timing islands, each running a separate clock. In many instances, the node with the fastest clock actually forms its own timing island.

A. Design

There are two major causes behind the failure of IEEE 802.11's clock synchronization mechanism when used in multi-hop wireless ad hoc networks. First, because beacon frames from nodes in a collision domain compete for the channel access on an equal footing, the probability that the

node with the fastest clock in a collision domain wins in this competition actually decreases when the number of nodes in the domain increases. This explains why the effectiveness of IEEE 802.11's clock synchronization mechanism deteriorates with the number of nodes in a multi-hop network. Second, when a wireless network node receives a beacon frame, it replaces its TSF counter with the beacon's timestamp if the timestamp is larger than the current TSF counter value, but *always* cancels the pending beacon timer. That is, beacon frames scheduled to be transmitted from nodes with a faster clock may be suppressed by those from nodes with a slower clock. Moreover, in every beacon period, there will be *only one* node in each collision domain that successfully sends out its beacon. Consequently, either a beacon frame from the node with the fastest clock does not have a chance to be transmitted at all, or even when it is successfully sent out, it cannot be successfully relayed to all the other nodes.

Because IEEE 802.11's clock synchronization mechanism is implemented in hardware, it avoids several sources of clock synchronization errors typically associated with software-only clock synchronization implementations. First, because the timestamp carried by a beacon frame is attached at the time when the frame is about to be *physically* transmitted, all the MAC-related delay associated with a beacon frame, e.g., random back-off and retransmission will not contribute to the clock synchronization error. Second, when a beacon frame arrives at a receiving node, the network interface hardware directly updates the TSF counter if necessary, without involving any software running on the CPU. As a result, the synchronization error due to interrupt delivery latency and interrupt processing overhead does not exist in IEEE 802.11's clock synchronization mechanism. For these reasons, we strive to exploit IEEE 802.11 hardware's beacon transmission and processing capability as much as possible.

Ideally, if the clock rate of every node in a multi-hop wireless network is known, one can organize them into a tree in such a way that the root of the tree corresponds to the node with the fastest clock. Periodically the root sends out a beacon frame, and each node forwards a beacon to its children *immediately* after receiving it. In this scheme, because at any point in time only a small number of nodes in a neighborhood transmit beacon frames and beacon frames carry a higher priority than normal frames, the propagation of beacon frames down the tree incurs small latency and experiences few packet drops, and the resulting clock drift across the entire network is minimized.

To implement the ideal clock synchronization scheme described above on IEEE 802.11-based wireless interfaces, one needs to identify the node with the fastest clock in a given network and modify the beacon transmission logic so that each node sends out beacon frames at well-defined moments rather than at randomly chosen timer events. Unfortunately, identifying the node with the fastest clock in an arbitrary multi-hop network is expensive because it requires global coordination and because it needs to be invoked periodically to account for the fact that the node with the fastest clock may

change over time.

Recognizing the difficulty of identifying the node with the fastest clock in large networks, we come to a simple but radical alternative: instead of identifying from a given network the node with the fastest clock, we simply pick a node and *make* it the one with the fastest clock by jumping its TSF counter by a large amount. This *clock-jumping* approach completely bypasses the fastest node identification problem, and thus significantly reduces both the implementation complexity and the performance cost of clock synchronization.

More concretely, the proposed clock-jumping synchronization algorithm chooses a node in the network to be the root of the beacon distribution tree, and periodically increases its TSF counter's value by a sufficiently large amount that guarantees that its TSF counter value is larger than all other nodes in the beginning of every beacon period. The interval between two consecutive beacons with an artificially jumped timestamp is called a *jumping interval*.

In each beacon period, only the chosen root node *actively* sends out a beacon frame; all the other nodes *passively* wait for the root's beacon frame to arrive and forward the beacon further after a random-backoff delay. Because the root's TSF counter value is set up to be larger than any other node's TSF counter when it sends out a synchronization beacon, those receiving the beacon will update their TSF counter with the timestamp carried in the beacon by virtue of the original beacon processing logic built into IEEE 802.11-based wireless interfaces. Eventually every other node in the network is synchronized with the root node's TSF, even though its clock is not necessarily the fastest in the network.

In addition to being simple and efficient, the proposed algorithm also completely eliminates the problem of nodes with a slower clock suppressing beacon transmission on nodes with a faster clock, because by construction the beacons always flow from nodes with faster clock to nodes with slower clock. Finally, this algorithm does not require any explicit build-up of a distribution tree. Instead, a tree is naturally formed from the chosen root over the physical wireless connectivity graph.

B. Analysis

One problem with the proposed clock-jumping synchronization algorithm is that some relayed beacon frames may be lost because of collision and consequently some network nodes may fail to synchronize their TSF counters with the root's TSF counter. This is not a serious problem in practice for two reasons. Let the set of nodes that attempt to transmit beacon frames with the same jumped timestamp *simultaneously* be S_{beacon} . First, because S_{beacon} is in general a small subset of the entire network, the probability of beacon frame loss due to repeated collision is much lower than IEEE 802.11's original clock synchronization mechanism. Second, even if some members in S_{beacon} could not successfully send out their beacon frames, it is still unlikely that certain network nodes fail to receive the corresponding synchronization beacons as a result. If the nodes in S_{beacon} are sparsely populated, it is unlikely that they will collide with one another to the extent

that some of them eventually give up forwarding their beacon frames. If the nodes in S_{beacon} are densely populated, it is OK that some of the nodes fail to forward their beacon frames, because those successfully forwarded beacon frames may already be sufficient to bring the entire network to synchronize with the root. The experimental results in the next section confirm the above analysis.

Another problem with the proposed clock synchronization algorithm is that if the selected root node dies, the clock synchronization mechanism stops working. To address this issue, we need to solve the leader election problem, to which many possible solutions exist [14] [15] [16]. We chose a simple solution that fits well with the structure of the clock-jumping synchronization algorithm.

Assume at any point in time, every node in a multi-hop wireless network knows how many hops it is away from the root. This is achieved by keeping a hop count field in each beacon frame, and incrementing it by one every time it passes through a node in the beacon distribution tree. This hop count information also allows a node to ignore beacon frames forwarded by its siblings or descendants. Given the hop count information, every wireless network node sets a beacon timer with an initial time-out value that is larger than the jumping interval and is proportional to its recorded hop count. An expiration of a node's beacon timer means a beacon with a jumped timestamp should have arrived but did not, and the node assumes that the current root has died and starts to compete to be the new root. To prevent spurious beacon generation due to beacon frame loss, the root node sends every beacon frame that carries a jumped timestamp K times, where K is 2 or 3.

When the old root node of a beacon distribution tree dies, its one-hop neighbors notice this first. Then each of them generates a new beacon that carries a timestamp equal to the next expected jumped clock value, and waits for a random transmission delay before broadcasting it to its neighbors. When a node receives a beacon with a jumped timestamp, it cancels any pending beacon timer if any, forwards the beacon and considers itself as a child of the node sending/forwarding the beacon in each of the following cases:

- It has a pending beacon timer that has not expired,
- Its beacon timer has expired and its TSF is smaller than the beacon's timestamp, or
- It does not have any pending beacon timer

In the case that the receiving node's beacon timer has expired and its TSF counter is larger than the beacon's timestamp, the receiving node does nothing and continues to assume it is the new root. Although this design may result in multiple new roots temporarily, only the beacon from the one-hop neighbor with the smallest transmission delay prevails eventually, and that neighbor becomes the ultimate new root. While the above leader election algorithm was originally designed for wireless mesh networks, whose topology is largely fixed, it is equally applicable to mobile ad hoc networks, because topology change only increases the convergence time and the

default jumping interval (< 1 second) limits the possible extent of topology change in real-world mobile ad hoc networks.

C. Implementation Issues

The proposed clock-jumping synchronization algorithm entails two implementation challenges: (1) how to increase the value of a TSF counter on an IEEE 802.11 wireless interface through software, and (2) how to immediately relay a beacon frame after receiving it. These are challenging because the hardware specifications of most commercial IEEE 802.11 wireless LAN interfaces are proprietary and their drivers are close-sourced.

MadWifi [17] is a partially open-source Linux driver for IEEE 802.11 WLAN cards based on Atheros chips. Such WLAN cards contain a 64-bit micro-second resolution TSF counter, which the MadWifi driver exposes as two 32-bit registers, *MTSF* and *LTSF*, corresponding to the upper and lower 32 bits of the TSF counter. The MadWifi driver allows only *MTSF* to be modified, but not *LTSF*. As a result, the current implementation of the clock-jumping algorithm increments the 13th bit of *MTSF* and results in an increment of 2^{44} micro-seconds or about half a year, which we believe is sufficient to ensure that a new beacon's timestamp is larger than the TSF counter values of all the nodes in a typical multi-hop wireless network.

MadWifi supports multiple modes of beacon transmission. One of them is called *Virtual End Of List* (VEOL), which handles all beacon processing and transmission without generating any interrupt to the CPU. The VEOL mode is convenient because once it is properly set up, processing and transmission of all subsequent beacons are completely done by hardware without incurring any software overhead. However, the VEOL mode is inappropriate because it does not allow the software to control the beacon processing and transmission logic. Therefore, we choose another beacon mode in MadWifi called *Software Beacon Alert* (SWBA), which triggers an interrupt and invokes a software interrupt service routine before the transmission of each beacon. Inside this interrupt service routine, we implement the clock-jumping algorithm by modifying the TSF counter and scheduling a short timer to forward the beacon. To use the SWBA mode in the MadWifi driver, we need to disable its VEOL mode. However, disabling VEOL triggers a severe software bug inside the driver, but we eventually fixed this problem by applying several software patches and driver modifications.

D. Application Consideration

An ideal clock synchronization scheme should provide clock-based applications both monotonicity, where the clock value never decreases, and continuity, where the increment in the clock value is constant in each time unit. Note that perfect continuity is impossible as long as the clocks in a system are not synchronized. By definition, clock synchronization is meant to force these clocks to the same value and thus inevitably must introduce discontinuity to some clocks along the way. As long as the discontinuity a clock synchronization

scheme introduces is comparable to the maximal drift among the clocks, such discontinuity is considered acceptable.

Lets denote the m to n bits of a variable X as $X[m:n]$. An ideal implementation of clock-jumping algorithm is for the chosen root node to increase its TSF counter by MCD every time it jumps its clock, where MCD corresponds to the maximal clock drift in a given network and thus need to be customized to each network. However, because existing WLAN interfaces only allow the *MTSF* part of their TSF counter to be modified, this ideal implementation is not possible on current WLAN hardware interfaces. The ideal clock-jumping implementation requires the root node to increment the *MTSF* part of its TSF counter and expose the entire 64-bit TSF counter value to applications. This implementation supports monotonicity, but introduces significant discontinuity, which may trigger undesirable side effects for clock-based applications. A slight revision of this implementation, which exposes only the *LTSF* part of the TSF counter to applications, eliminates the discontinuity problem but introduces the problem of frequent wrap-arounds of the clocks .

To address this problem, the final clock-jumping implementation increases the *MTSF* of the root nodes TSF counter by 2^{12} every time when it jumps the clock. Moreover, whenever an application asks for the current value of the TSF counter, the clock-jumping driver returns $TSF[0:43] + MCD$. Essentially, only the least significant 44 bits of the TSF counter are exposed to applications. Because the TSF counter increments its value every μsec , a 44-bit range corresponds to 203 days, which is sufficiently long for most applications.

Suppose the value of the root nodes TSF counter is X at the time immediately before it is about to jump the clock. $X[0:43]$ remains the same after the clock is jumped and is embedded in the beacon frame to be distributed. When a node receives the beacon, it treats the actual clock value as $X[43:0] + MCD$. If the root node happens to be the slowest node in the network, $X[43:0] + MCD$ should still be larger than the $TSF[0:43]$ value of the fastest node, and thus guarantees monotonicity. If the root node happens to be the fastest node in the network, then the slowest node experiences a discontinuity of $2 * MCD$. Therefore, this scheme supports monotonicity, continuity and sufficiently long wrap-around period.

The only problem is that the most significant 20 bits of the TSF counter overflow and wrap-around frequently. Assume this 20-bit range is incremented once every clock-jumping interval, which is defaulted to 100 msec, then it will overflow every 29 hours. Fortunately, the clock-jumping implementation can hide the wrap-around of $TSF[44:63]$ from applications completely by resetting it in software every time it is about to wrap around. Note that although this reset operation modifies the *MTSF* part of the TSF counter, it preserves the $TSF[32:43]$ part, and therefore is transparent to applications.

IV. PERFORMANCE EVALUATION

A. Testbed Setup

Fig. 1 shows the physical testbed used in this performance study. There are five computers connected to an Ethernet

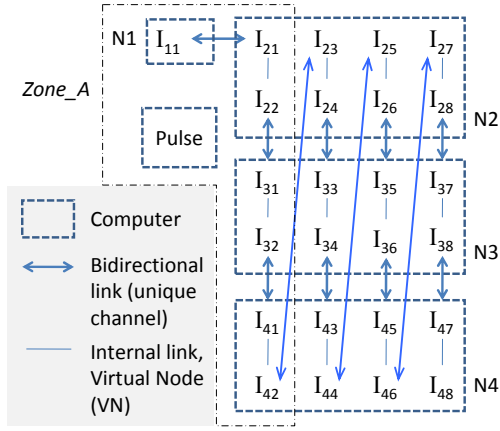


Fig. 1. Using a multi-channel single-hop physical network to emulate a multi-hop single-channel logical network. Each virtual node in the emulated network consists of two WLAN interfaces. The bidirectional bold arrow represents a wireless link between two virtual nodes whereas each lightly shaded line connects the two WLAN interfaces associated with the same virtual node.

switch, N_1 , N_2 , N_3 , N_4 and Pulse. Each of them is a Dell desktop machine with a Pentium-4 1.5-GHz CPU, 256 Mbytes of RAM and an 100Mbps Ethernet interface. N_1 contains a single WLAN interface while N_2 , N_3 and N_4 each have 8 WLAN interfaces. All WLAN interfaces are Wistron NeWeb 802.11 a/b/g mini-PCI card (Model No. CM9), which are installed on a computer via a 4-port miniPCI-to-PCI adaptor. The NeWeb 802.11 a/b/g mini-PCI card uses the AR5004X chipset, which in turn consists of an AR5213 MAC controller chip and an AR5112 dual-band radio. All computers run the Linux 2.4.26 kernel and the MadWifi driver 0.9.3.1. The default beacon period and jumping interval of all WLAN interfaces is set to **100 msec** if not otherwise specified.

Because it is difficult to set up a true *single-channel multi-hop* IEEE 802.11-based network within a limited physical space, we set up a *single-hop multi-channel* network to approximate it. Let $I_{i,j}$ denote the j -th interface of the computer N_i . As shown in Fig. 1, each pair of WLAN interfaces forms a virtual node (VN) in the approximated multi-hop network. For example, I_{21} and I_{22} form one VN, I_{31} and I_{32} form another VN, I_{23} and I_{24} another VN, etc. In addition, a distinct non-overlapped radio channel is assigned to each wireless link between neighboring VNs so that these wireless links do not interfere with one another even if they belong to the same collision domain. The dark arrows in Fig. 1 indicate these inter-VN wireless links. Therefore Fig. 1 represents a 13-node 12-hop wireless network with a linear-chain topology.

To test the clock-jumping synchronization algorithm on this emulated multi-hop wireless network, we ran the following experiment. Periodically, a beacon with a jumped timestamp is sent out from the interface $I_{1,1}$. Every time an interface $I_{i,j}$, where $i \in \{2, 3, 4\}$ and $j \in \{1, 3, 5, 7\}$, receives a beacon with a jumped timestamp, it copies the timestamp to its TSF, copies the TSF's upper 20 bits to that of the interface $I_{i,j+1}$ and schedules an immediate beacon transmission on $I_{i,j+1}$. Note that the lower 44 bits of the TSFs of the two WLAN interfaces

on a VN are not synchronized, because these bits cannot be set explicitly by software. Therefore, only the upper 20 bits of the original jumped timestamp on $I_{1,1}$ are propagated through the network.

Given that the the lower 44 bits of the two TSFs on each VN are not synchronized, it is not clear how to measure the clock drift among the VNs in the emulated multi-hop network. Fortunately, a more careful analysis suggests that meaningful clock drift measurement is still possible for the following reasons:

- The global clock drift across a linear-topology network can be derived if the relative clock drift between each pair of immediately adjacent network nodes is known. Consider a simple linear-topology network of three nodes, A, B, and C, where A is connected to B, which is then connected to C. Let $Drift_{X,Y}$ be the clock drift between X and Y. If $Drift_{A,B}$ and $Drift_{B,C}$ are known, then $Drift_{A,C}$ can be easily derived by adding $Drift_{A,B}$ to $Drift_{B,C}$. With this method, we can compute the relative clock drift between a reference node and every other node in the network. These computed clock drifts could be either positive or negative. Then the global clock drift of a network is equal to $(Drift_{Max} - Drift_{Min})$, where $Drift_{Max}$ and $Drift_{Min}$ represent the maximum and minimum of these relative clock drifts, respectively. This idea can be extended to an arbitrary-topology N-node network by mathematical induction. From this point on, we will refer to the relative clock drift between the i -th interface and the reference interface as $Drift_i$.
- The relative clock drift between two adjacent VNs in our emulated testbed can be estimated even when the two TSFs on each VN are not fully synchronized. Consider the interfaces I_{21} , I_{22} and I_{31} in Fig. 1, where I_{21} and I_{22} belong to one VN and I_{31} belongs to an adjacent VN. Let's call the TSF values on these three interface at a particular point in time as $Clock_{21}$, $Clock_{22}$, and $Clock_{31}$. During beacon frame propagation, the upper 20 bits of $Clock_{21}$ are propagated to the upper 20 bits of $Clock_{22}$, and the entire 64 bits of $Clock_{22}$ are propagated to $Clock_{31}$. Therefore, the relative clock drift between these two VNs is determined by the difference between $Clock_{22}$ and $Clock_{31}$, but has nothing to do with the difference between $Clock_{21}$ and $Clock_{22}$. Consequently, by subtracting I_{22} from I_{31} , we can readily measure the relative clock drift between these two VNs, regardless of whether $Clock_{21}$ and $Clock_{22}$ are the same or not.

B. Measurement of Inherent Global Clock Drift

To evaluate the effectiveness of the proposed clock-jumping synchronization algorithm, it is essential to first establish quantitatively the extent to which the TSF counters on commodity IEEE 802.11-based WLAN interfaces become out of synchronization over time when they are not explicitly synchronized. Towards that end, we set up a separate computer, Pulse in Fig. 1, to send out UDP broadcast packets, each carrying a 2-byte sequence number, on the wired Ethernet link once every

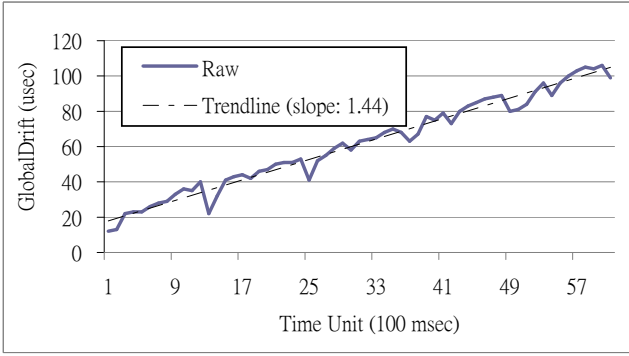


Fig. 2. The inherent global clock drift in a 3-hop linear-topology network that disables the synchronization mechanism on their WLAN interfaces. The Raw line shows the global clock drift measurements whereas the Trendline line represents the smoothed trend of the Raw line.

100 msec. Whenever each of the other computers receives a UDP broadcast packet from its Ethernet interface, it records the current values of the TSF counters on its WLAN interfaces at that instant and the sequence number of the received packet. Then we collect all the records of TSF counter values and sequence numbers from all the computers and perform an off-line analysis of them. Essentially we use the receiving times of these UDP broadcast packets as reference points when comparing the TSF counter values on the virtual nodes.

Let C_τ be a TSF counter's value at time τ , and C_{start} be the first recorded value of the TSF counter. Then $ClockAdv(\tau)$, which is defined as $C_\tau - C_{start}$, represents the number of ticks passed between when the first broadcast packet is received and the time τ . Therefore, the relative clock drift between the i -th WLAN interface and the reference WLAN interface, $Drift_i$ in the period between when the first broadcast packet is received and the time τ is $ClockAdv_i(\tau) - ClockAdv_{reference}(\tau)$, and the global clock drift across the network during the same period is

$$GlobalDrift(\tau) = Drift_{max}(\tau) - Drift_{min}(\tau) \quad (1)$$

where $Drift_{max}(\tau)$ and $Drift_{min}(\tau)$ represent the maximum and minimum of all $Drift_i(\tau)$'s, respectively. Finally, $AvgDrift$ and $MaxDrift$ are defined as the average and maximum of all $GlobalDrift$ values collected in the experiment runs.

To quantify the inherent clock drift among commodity IEEE 802.11 interfaces, we picked a subset of machines in the testbed, marked as Zone A in Fig. 1, to emulate a 3-hop 4-node linear-topology network, and turn off beacon frame transmission on their WLAN interfaces. Because clock synchronization is disabled, the TSF counters of these interfaces keep drifting apart over time, and $GlobalDrift$ is expected to increase linearly with the elapsed time. However, the measurements of $GlobalDrift$, shown as the Raw line in Fig. 2, do not form a monotonically increasing line but actually exhibit a zigzag form with minor ups and downs, because the *synchronization measurement error* associated with the Ethernet broadcasting approach is much larger than and thus over-shadows the clock

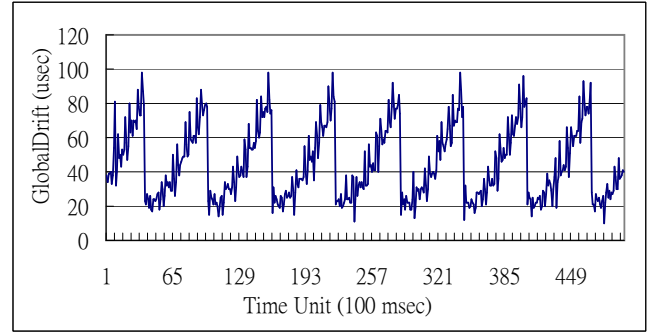


Fig. 3. The global clock drift of the clock-jumping algorithm in a 3-hop linear-topology network. The beacon interval and jumping interval are both set to 6400 msec.

drift among the testbed nodes. More concretely, the delay from when a UDP broadcast packet reaches a receiver node's Ethernet interface to when it successfully records its WLAN interface's TSF counter value could vary significantly from one node to another and even from one packet to another on the same testbed node.

To determine the clock drift per unit time (100 msec in this figure) or *clock drift rate*, we compute the smoothed trend of the Raw line, shown as Trendline in Fig. 2. The slope of this smoothed trend line is 1.44, and represents the clock drift rate. In this case, 1.44 means the global clock drift of the 4-node test network increases by 1.44 μ sec every 100 msec. To demonstrate that this is indeed the inherent clock drift rate, we applied the clock-jumping algorithm to the 4-node test network with node A as the clock reference node, and the beacon and jumping intervals both set to 6400 msec.

Fig. 3 shows the expected result: $GlobalDrift$ rises over time with a slope of around 1.44, but sharply declines in every jumping interval because the clock-jumping algorithm synchronizes every test network node's TSF at the beginning of every jumping interval.

Suppose the measured $GlobalDrift$ when $X = 1$ in Fig. 2 is z , then the synchronization error of Ethernet broadcasting is roughly $z - 1.44$. Intuitively the area under the $GlobalDrift$ curve and above the $Y = z - 1.44$ line is a good measure of the total clock drift. This area can be approximated by $AvgDrift$, which is the average of all measured $GlobalDrift$ values minus the synchronization measurement error associated with Ethernet broadcasting. Therefore, $AvgDrift$ is a good metric to quantitatively evaluate the effectiveness of a clock synchronization algorithm on a multi-hop wireless network. On the other hand, $MaxDrift$ is less useful because it tends to be distorted and dominated by the synchronization measurement error on the wired Ethernet, especially when the beacon period is set to be small.

C. Clock Synchronization Accuracy

To measure the $AvgDrift$ of the proposed clock jumping-based synchronization algorithm on a multi-hop wireless network, we used the setup shown in Fig. 1 to construct a linear multi-hop network with 4 to 10 hops. The beacon and jumping

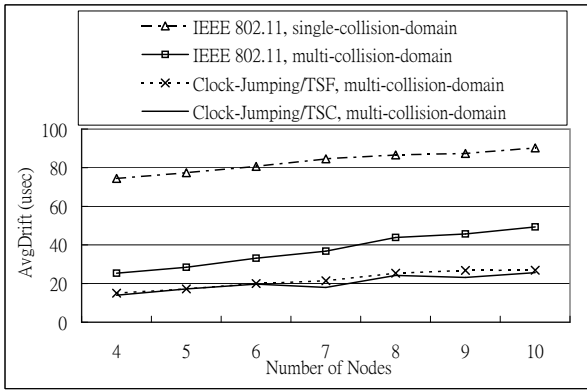


Fig. 4. The average clock drift of a multi-collision-domain linear-topology network running IEEE 802.11 TSF, clock-jumping algorithm based on TSF, and clock-jumping algorithm based on TSC, and of a single-collision-domain linear-topology network running IEEE 802.11 TSF. In all cases, the average clock drift increases with the number of nodes in the network. The number of nodes is equal to the number of hops in a multi-collision-domain linear network.

interval are both set to default 100 msec, and their measured *AvgDrift*s are shown as the line “*Clock-Jumping/TSF, multi-collision-domain*” in Fig. 4. As expected, *AvgDrift* increases with the number of hops in the network for two reasons. First, because it takes longer for a new jumped timestamp to propagate from the root to the leaves of the beacon distribution tree when the diameter of the network is larger, *AvgDrift*, which reflects the degree to which a network’s nodes stay synchronized with one another, grows slightly with the number of hops in the linear test network. Second, on the testbed each physical computer supports 8 IEEE 802.11 WLAN interfaces and up to 4 virtual nodes, and it takes certain CPU time to retrieve and log all the TSF counters every time it receives a UDP broadcast packet on the Ethernet interface. As a result, the more hops in the linear test network, the more TSF counter values each physical computer needs to access upon receiving a UDP broadcast packet, and the less accurate the recorded values of those TSF counters are. By subtracting from the recorded value of a TSF counter the amount of time required to access all the TSF counters before it, we effectively eliminated synchronization errors due to this experimental artifact.

Instead of the TSF counter on the wireless LAN interface, one can also use the performance counter on modern CPUs as the basis for a general clock synchronization service. In the X86 architecture, the CPU’s Time Stamp Counter (TSC) could play this role. Every time a TSF counter on a node is updated by a beacon frame, we also record its TSC counter’s value. For multi-hop network, the *AvgDrift* result based when the TSC counter is used, are shown as the line “*Clock-Jumping/TSC, multi-collision-domain*” in Fig. 4. When compared with the TSF results, TSC reduces *AvgDrift* by 5%. This suggests that the TSC counter has less drift, presumably because the oscillator in the CPU runs at a higher clock frequency than that on the IEEE 802.11 WLAN interface.

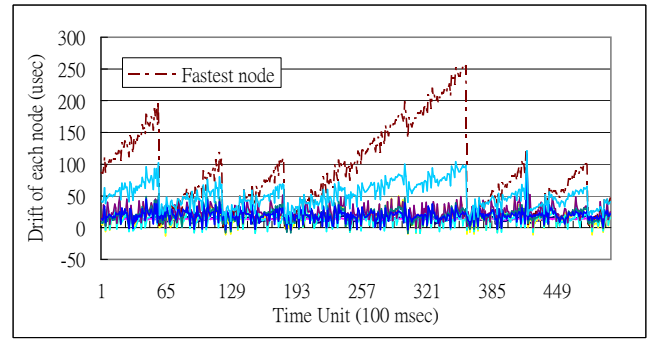


Fig. 5. The temporal evolution of the clock drift of each node in a 10-node linear-topology network with a 6400-msec beacon interval. The fastest node’s clock drift does not return to zero in every beacon interval.

D. Comparison with IEEE 802.11’s Clock Synchronization Mechanism

We used the same setup in the previous subsection to compare the *AvgDrift* of the proposed clock-jumping synchronization algorithm with IEEE 802.11’s native clock synchronization mechanism when its beacon interval is set to 100 msec. Marked as “*IEEE 802.11, multi-collision-domain*” in Fig. 4, the *AvgDrift* of IEEE 802.11’s native clock synchronization algorithm is 66% higher than that of the clock-jumping algorithm when the number of hops is 4, and is 81% higher when the number of hops is 10. This suggests that not only the clock-jumping algorithm is more accurate than IEEE 802.11’s native clock synchronization mechanism, but also their gap increases with the number of hops in the linear test network. To understand why, we investigate the two known problems of IEEE 802.11’s clock synchronization mechanism in more detail using the following two experiments.

a) **Fastest Node Asynchronism:** From the analysis of Huang and Lai [2], the probability for the fastest node to successfully send out a beacon frame decreases as the number of nodes increases in the network, because the fastest node has to compete with more nodes in channel access. We set up a single-hop network with all the nodes in a single collision domain with a 6400 msec beacon interval, and measure the *AvgDrift* when the number of nodes increases from 2 to 10. The *AvgDrift* curve corresponding to “*IEEE 802.11, single-collision domain*” in Fig. 4 keeps increasing as the number of nodes in the collision domain grows. To show that the fastest node asynchronism problem does exist, we further examine the temporal evolution of the clock drift for each node in a 10-node network. In Fig. 5, the fastest node in the network is the one whose clock drift is the largest, and from time to time its clock drift is brought down to zero because it is synchronized with the reference. However, the fact that the fastest node’s clock drift does not return to zero in every beacon interval (6400 msec) indicates that its clock does often drift away from the rest of the network, sometimes for up to 3 beacon intervals. This result empirically demonstrates the existence of the fastest node asynchronism problem in an IEEE 802.11 network.

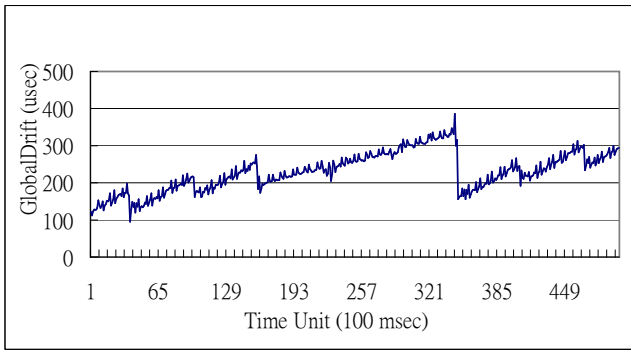


Fig. 6. The global clock drift of IEEE 802.11's clock synchronization mechanism in a 3-hop linear-topology network whose beacon interval is set to 6400 msec.

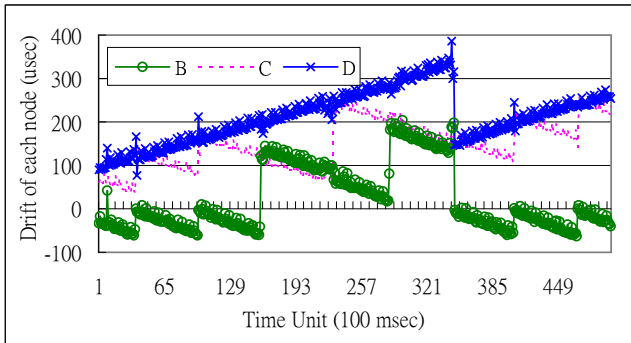


Fig. 7. The temporal evolution of the clock drift of each node in a 4-node linear-topology network with a 6400-msec beacon interval. Nodes form multiple clock islands, and the island formation changes over time.

b) **Time Partitioning:** To investigate the time partitioning problem, where a network is divided into multiple disjoint clock islands, we ran IEEE 802.11's clock synchronization mechanism on a 3-hop linear test network with 6400 msec beacon interval whereas each node is named A, B, C, and D in topological order. The evolution of the test network's *GlobalDrift* over time is shown in Fig. 6. Compared with Fig. 3, the test network's *GlobalDrift* in Fig. 6 does not drop in value in every beacon interval. This means that IEEE 802.11's clock synchronization mechanism cannot always bring the entire network to synchronization using a single beacon frame.

To demonstrate that the time partitioning problem does exist, we show the evolution of the relative clock drift of B, C and D with respect to A over time in Fig. 7. Through calibration, we know the clock of D is the fastest, followed by the clocks of A, B and C in decreasing order. Between time 0 and time 153, there are two clock islands: C is synchronized with D and B is synchronized with A. At time 153, B joins the island of C and D, and A is on its own. At time 347, B rebinds with A but is separated from the island of C and D, and again there are two islands each with 2 nodes. Although C's clock seems to be following D's clock, it is not synchronized with D in every beacon interval, as indicated by the fact that C's curve does not follow a perfect periodic zigzag pattern.

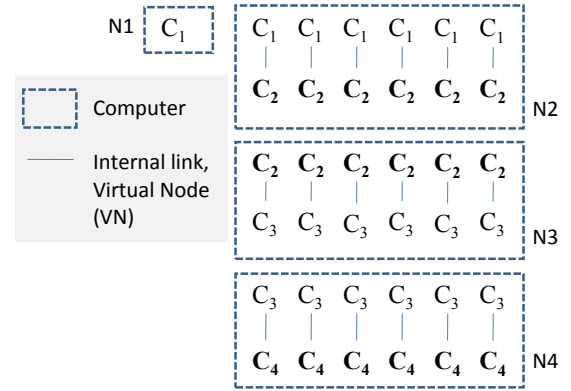


Fig. 8. A tree-topology network that simulates a 19-node 3-hop network with C1/N1 as the synchronization root. C_i refers to a WLAN interface that operates at channel i whereas each lightly shaded line connects the two WLAN interfaces associated with the same virtual node.

In summary, the proposed clock-jumping synchronization algorithm is better than IEEE 802.11's clock synchronization algorithm in terms of not only average clock drift, but also maximum clock drift. Therefore, it is a more robust clock synchronization mechanism for multi-hop wireless networks whose worst-case clock synchronization error must be bounded.

E. Impact of Beacon Frame Loss on Clock-Jumping Algorithm

To evaluate the impact of beacon frame loss, for example due to repeated collision, on the effectiveness of the proposed clock-jumping synchronization algorithm, we modified the testbed as Fig. 8 so that N2, N3 and N4 each carries 12 WLAN interfaces or 6 virtual nodes, the N3-facing WLAN interfaces on N2 and the N2-facing WLAN interfaces on N3 are assigned the same radio channel, and the N4-facing WLAN interfaces on N3 and the N3-facing WLAN interfaces on N4 are assigned the same radio channel. This configuration effectively simulates a 19-node 3-hop network with C1/N1 as the synchronization root and each level of the beacon distribution tree has 6 virtual nodes. Although the probability of collision for beacon frame forwarding is higher, the *AvgDrift* for this test network is about 15 μ sec, which is roughly the same as the *AvgDrift* measured on a 4-node 3-hop linear test network. The results of this experiment demonstrate that beacon frame loss indeed has no impact on the clock-jumping algorithm's effectiveness in practice, and that the clock-jumping algorithm works equally effective for networks with a general non-linear-chain topology.

F. Impact of Root Failure on Clock-Jumping Algorithm

To demonstrate the effectiveness of the root failure recovery mechanism in the proposed clock-jumping algorithm, we run the clock-jumping algorithm on the same 19-node 3-hop tree-topology network with high collision probability. Some time after the experiment run starts, as indicated by the dark triangle in Fig. 9, we unplug the power cord of the root node, and measure the global clock drift of this test network over time. Because of the death of the original root, the remaining nodes

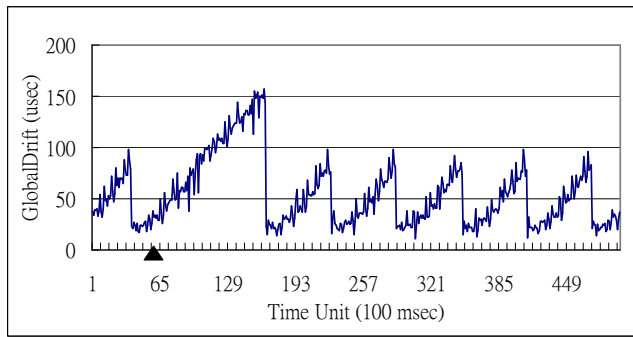


Fig. 9. When the root of a 3-hop linear-topology network dies at the time indicated by the triangle, the global clock drift keeps increasing for two beacon intervals and then be brought down because the root's neighbor takes over as the new root.

miss the next synchronization event immediately after the root's death. However, the immediate neighbor of the initial root, notices the death through its beacon timer, whose time-out value is set to twice the value of the beacon interval in this case, and sends out a new beacon with the next expected jumped stamp. As a result, the entire network is brought to synchronization within 2 beacon intervals.

V. CONCLUSION

Clock synchronization is an important building block for many wireless protocols, services and applications. The IEEE 802.11 standard provides a clock synchronization mechanism for both infrastructure-mode and ad hoc-mode operation. Unfortunately, its ad hoc-mode version does not work very well because every wireless network node can send out beacon frames and the beacon frames from nodes with a faster clock do not have a higher probability of winning in radio channel access than those from nodes with a slower clock. This deficiency leads to two well known problems: fastest node asynchronism and time partitioning. Both problems are worsened as the number of collision domains (or hops) and the number of nodes in each collision domains in the network increases.

In this paper, we present the design, implementation and evaluation of a novel *clock-jumping* approach to synchronizing the nodes in an IEEE 802.11-based multi-hop wireless network. This algorithm is not only simpler but also more accurate and efficient than IEEE 802.11's native clock synchronization mechanism, because it enables the fastest node to start a beacon-based synchronization transaction across the entire network, thus eliminating the problem that faster nodes may not be able to send/relay beacon frames by construction. In addition, the proposed algorithm exploits the hardware-based TSF counter update mechanism in IEEE 802.11-based WLAN interfaces, thus reducing the software-induced synchronization error to the minimum. More specifically, this work makes the following research contributions:

- A novel *clock-jumping synchronization* algorithm, which has successfully been implemented on IEEE 802.11 WLAN interfaces without requiring any

hardware/protocol modifications, and achieves the best synchronization accuracy on multi-hop IEEE 802.11-based networks while preserving the monotonicity and continuity of clock values, when compared with other wireless network clock synchronization schemes reported in the literature

- An innovative experimentation methodology that allows one to accurately effectively measure the clock drift of a single-channel multi-collision-domain wireless network using a single-collision-domain multi-channel wireless network.
- An empirical study of the effectiveness of IEEE 802.11's native clock synchronization mechanism on multi-hop wireless networks, and the extent of its deficiencies. All previous such studies are based on simulations.

REFERENCES

- [1] *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification*, IEEE standard 802.11, 1999.
- [2] L. Huang and T.-H. Lai, "On the scalability of ieee 802.11 ad hoc networks," *ACM MobiHoc*, June 2002.
- [3] D. Zhou, L. Huang, and T.-H. Lai, "On the scalability of ieee 802.11 ad-hoc-mode timing synchronization function," *ACM Wireless Networks*, August 2008.
- [4] D. L. Mills, "Internet time synchronization: the network time protocol," *IEEE Trans. Commun.*, October 1999.
- [5] F. Guo and Tzi-cker Chiueh, "Software tdma for voip applications over ieee802.11 wireless lan," *IEEE INFOCOM*, 2007.
- [6] F. Guo and Tzi-cker Chiueh, "Comparison of qos guarantee techniques for voip over ieee802.11 wireless lan," *Proceedings of the 15th Annual Multimedia Computing and Networking Conference (MMCN 2008)*, January 2008.
- [7] J. So and N. Vaidya, "Mtsf: A timing synchronization protocol to support synchronous operations in multihop wireless networks," *Technical Report, UIUC*, October 2004.
- [8] J.-P. Sheu, C.-M. Chao, and C.-W. Sun, "A clock synchronization algorithm for multi-hop wireless ad hoc networks," *International Conference on Distributed Computing Systems*, 2004.
- [9] Q. Li and D. Rus, "Global clock synchronization in sensor networks," *IEEE INFOCOM*, 2004.
- [10] J. Elson, L. Girod, and D. Estrin, "Fine-grained network time synchronization using reference broadcasts," *Fifth Symposium on Operating Systems Design and Implementation (OSDI)*, December 2002.
- [11] S. Ganeriwal, R. Kumar, and M. B. Srivastava, "Timing-sync protocol for sensor networks," *First ACM Conference on Embedded Networked Sensor Systems (SenSys)*, November 2003.
- [12] D. Veitch, S. Babu, and A. Psztor, "Robust synchronization of software clocks across the internet," in *Proc. ACM SIGCOMM Internet Measurement Conference*, October 2004.
- [13] E. Corell, P. Saxholm, and D. Veitch, "A user friendly tsc clock," in *Passive and Active Measurement Conference (PAM2006)*, March 2006.
- [14] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "Energy-efficient communication protocol for wireless microsensor networks," in *Proc. of the 33rd Hawaii International Conference on System Sciences*, 2000, pp. 1–10.
- [15] O. Younis and S. Fahmy, "Heed: A hybrid, energy-efficient, distributed clustering approach for ad-hoc sensor networks," *IEEE Trans. Mobile Comput.*, 2004.
- [16] L. Zhao, X. Hong, and Q. Liang, "Energy-efficient self-organization for wireless sensor networks: a fully distributed approach," *Global Telecommunications Conference*, 2004.
- [17] Madwifi project, "The madwifi driver for atheros-based wireless lan interfaces," <http://madwifi.org>.