Understanding Precision Time Protocol in Today's Wi-Fi Networks: A Measurement Study

Paizhuo Chen and Zhice Yang, *ShanghaiTech University*

https://www.usenix.org/conference/atc21/presentation/chen

This paper is included in the Proceedings of the 2021 USENIX Annual Technical Conference.

July 14–16, 2021

978-1-939133-23-6

Understanding Precision Time Protocol in Today's Wi-Fi Networks: A Measurement Study

Paizhuo Chen Zhice Yang
School of Information Science and Technology, ShanghaiTech University

Abstract

Emerging mobile applications involving distributed control and sensing call for accurate time synchronization over wireless links. This paper systematically studies the performance of Precision Time Protocol (PTP) in today's Wi-Fi networks. We investigate both software and hardware PTP implementations. Our study uncovers the root causes of software PTP synchronization errors. We show that with fine-tuned system configurations and an online calibration procedure, software PTP can achieve reasonable accuracy with off-the-shelf Wi-Fi devices. Hardware PTP requires a PTP hardware timestamper clock not contained in Wi-Fi NICs. We propose a method to make use of the hardware TSF counter to emulate the PTP clock. Rigorous tests traversing various conditions show that both software and hardware PTP implementations can achieve a 1-μs level of accuracy in today's Wi-Fi networks.

1 Introduction

Emerging mobile devices are rich in sensing, actuation, and computational capabilities. A shared time basic is essential for coordinating multiple of these devices and fusing the data they collect to enable innovative applications [23, 44]. For example, if the cameras of multiple co-located smartphones are synchronized, the videos they record can be used for 3D view synthesis, which can shift the production of volumetric [20] and free viewpoint [12] videos from fixed and professional studios to flexible and amateur scenarios; If drones and robots are synchronized, a fleet of them can cooperate, *e.g.*, to deliver a large object exceeding their individual capacities.

There are many ways to wirelessly synchronize (sync) mobile devices. Some are laboratory prototypes [31, 41] while some are already used in commercial products, *e.g.*, GPS [38], cellular [46]. They differ in targeting accuracy and application scenarios. For general networked devices, a convenient solution is through network protocols. In this paper, we focus on synchronization in Wi-Fi networks, as Wi-Fi is prevalent and will continue to play a critical role in future edge networks.

There are several synchronization choices in Wi-Fi networks. IEEE 802.11 itself defines the Time Synchronization Function (TSF) [3], which syncs clients to the connected AP

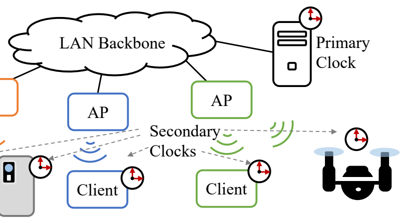


Figure 1: Scenarios of Wi-Fi Synchronization. APs are synchronized by the backbone. System clocks of Wi-Fi clients (*e.g.*, smartphones and drones) across different APs are synchronized to the primary clock. (In the context of this paper, we use Primary/Secondary clocks to replace the Master/Slave clocks used in the conventional synchronization terminology)

by resetting clients' time to the time broadcasted in the AP's beacons. However, TSF is only effective within a single AP's coverage, whereas the mobile clients may span a larger geometrical area. For example, the logistics robots might operate across multiple APs to transport items. To achieve synchronization on such a scale (Figure 1), the industry's standard practice is to use the Precision Time Protocol (PTP).

PTP is originally designed for LANs (IEEE 1588) [1] and can achieve nanosecond accuracy through timestamping the network packets. Attempts to extend PTP to wireless LAN or Wi-Fi can be traced back to the early Wi-Fi era [29]. The major difference from the wired situation is that the wireless channel introduces uncertainties in PTP timestamps [22]. This problem can be well addressed by hardware-assisted timestamping, *i.e.*, hardware PTP. The representative hardware PTP design is the latest IEEE 802.11 feature - Fine Timing Measurement (FTM) [3]. An alternative way is software PTP, which timestamps packets in the host software and hence is subject to various uncertainties caused by packet loss, channel contention, and host scheduling [22, 33]. Existing studies suggest ways to eliminate the uncertainties, *e.g.*, taking timestamps in the interrupt service routine (ISR) [33].

However, when we tried to apply the above implementations and insights to practical use, we encountered multiple friction and glitches, and finally failed. We intended to use

PTP to improve the synchronization of mobile cameras [20] for better 3D video synthesis. Our cameras are built on the Jetson Kit [14], a typical commercial off-the-shelf (COTS) development platform. For hardware PTP, while FTM seems perfect, we found that commercial FTM WNICs are mainly for positioning purposes [27] and the synchronization interface is proprietary and not publicly available. We then implemented software PTP based on existing work, but its performance is not as good as described. We found most of the related work was conducted around a decade ago, and thus many findings are no longer sufficient to explain and improve the PTP performance on current mobile CPUs and Wi-Fi NICs. To fulfill our application goal, we were forced to revisit PTP under the context of today's Wi-Fi networks and devices. We sought to uncover the factors affecting the PTP performance and ways to contain them.

A working PTP system mainly consists of two parts. The upper part, *i.e.*, the PTP daemon, such as `linuxptp` [15] and `gptp` [8], implements the PTP protocol state machine and clock tuning algorithms. The lower part timestamps PTP packets, and is finished in a way to minimize any timing uncertainties. The lower part, to our knowledge, only has proprietary implementations [11]. We develop two versions of the PTP lower part. The software PTP implementation timestamps packets in the WNIC driver. The hardware PTP implementation uses the TSF counter to timestamp packets in the WNIC hardware. They are both based on hacking and modifications on an open source WNIC driver and work seamlessly with existing PTP daemons. Their source code is available at [18]. Based on those, we conduct measurements to understand PTP performance in mobile platforms and practical Wi-Fi situations. Our key findings are:

• The major causes of software PTP's unstable performance are the power saving and performance optimization mechanisms employed in modern mobile systems. We reveal that WNIC's interrupt mitigation and CPU's idle power management are the dominating factors.

• With proper configurations, software PTP is reasonably accurate (sub-μs bias) and stable (1-μs jitter). A limitation of software PTP is the asymmetry interrupt responsiveness of host platforms. We propose a convenient online calibration method to address it.

• Hardware PTP can be supported by COTS WNICs by making full use of TSF counter. Although TSF counter is periodically affected by Wi-Fi TSF synchronization, our analysis suggests that it is still sufficiently accurate when the beacon frequency is as high as the default value.

• Hardware PTP is accurate, stable (sub-μs bias error and sub-μs jitter), and independent of software configurations. Similar to the wired PTP situation, calibration of the hardware timestamper is needed to achieve high accuracy when using different models of WNIC.

Our measurements traverse various workloads and conditions, covering both normal and extreme use scenarios. Our analysis provides a sound and up-to-date understanding of Wi-Fi PTP performance. The implementation incorporating our findings renders an accurate and scalable synchronization solution for general mobile systems. It allows us to continue investigating 3D video synthesis from videos captured by multiple mobile cameras.

2 Background

2.1 Clocks in Computer

A clock is a tool that measures the elapse of time, and normally consists of an oscillator and a counter. The oscillator periodically generates ticks at a fixed rate of R Hz. The counter is triggered on each tick and counts the number of ticks from the time it is powered on, says T_0. Let n denote the value of the counter, and n/R measures the elapsed time since T_0.

In practice, T_0 is arbitrary. It is more convenient to count the elapsed time T_{off} from a chosen reference time T_{ref}. T_{off} can be measured by other clocks. At time t_i, the counter of the clock is n_i, and the "time" of the clock is defined by

$$T_i = T_{\text{off}} + \frac{n_i}{R}. \quad (1)$$

T_i measures the elapsed time since T_{ref}. In particular, when T_{ref} is a known absolute time, *e.g.*, the start of 1 January 1970, we say T_i measures the absolute time. Equation (1) maps the value of the clock counter to the time of the clock through two parameters, $\{T_{\text{off}}, R\}$. This tuple uniquely determines the time of a clock.

Modern computers contain multiple clocks that belong to different hardware modules. This paper is related to three clocks: SYStem clock from the host computer, TSF clock from the WNIC, and PTP clock from the Ethernet NIC. Each clock has a counter, an oscillator, and the corresponding tuple $\{T_{\text{off}}, R\}$. System clock represents the clock that the software can refer to. In our experiments, TSC clock [10] in x86 PCs and CCNT clock [6] in ARM platforms are used as the source of system clock. We use T_i^{SYS} to denote the time of the system clock, *i.e.*, system time. When describing multiple hosts, we use an intermediate superscript to differentiate, *e.g.*, $T_i^{P_{\text{SYS}}}$ and $T_i^{S_{\text{SYS}}}$ denote the system time of clock P and clock S respectively.

TSF is defined in the IEEE 802.11 standard as a mandatory feature. Every WNIC maintains a 64-bit TSF counter with a 1 MHz tick rate. TSF counter can be accessed by software through reading TSF registers. We use n_i^{TSF} to denote the counter value, and T_i^{TSF} to denote the mapped TSF time. Similarly, T_i^{PTP} denotes the time of the hardware clock of the Ethernet NIC supporting PTP protocol.

2.2 PTP Basics

PTP is defined in the IEEE 1588 standard [1] to achieve time synchronization for networked computers on the LAN scale. PTP works by timestamping PTP packets. The protocol syncing the secondary clock (S) to the primary clock (P) is explained in Figure 2. The `SYNC` packet is sent out from the host of the primary clock, and timestamped by the primary clock as T_1^P. The SYNC packet is received by the host of the secondary clock, and timestamped by the secondary clock as T_2^S. In the reverse direction, the DELAY REQUEST packet triggers two timestamps T_3^S and T_4^P. FOLLOW UP and DELAY REPLY packets are used to convey T_1^P and T_4^P to the host of the secondary clock for calculating the clock offset. If the clock drift is ignored in the period of measurement, the four timestamps have the following relation: $T_1^P - T_2^S = \Delta - T_{\text{delay}}^{P\to S}; T_4^P - T_3^S = \Delta + T_{\text{delay}}^{S\to P}$, where T_{delay} denotes the network latency between the two hosts. In a single hop network connection like Wi-Fi, this value is stable and symmetric [1], *i.e.*, $T_{\text{delay}}^{P\to S} = T_{\text{delay}}^{S\to P}$, the offset from the primary clock can be calculated by

$$\Delta = \frac{(T_1^P - T_2^S) + (T_4^P - T_3^S)}{2}. \tag{2}$$

According to this offset value, the secondary clock can be adjusted, *e.g.*, by tuning $\{T_{\text{off}}^S, R^S\}$, to reduce any future Δ.

In practice, the synchronization accuracy of PTP depends on how precisely the four timestamps are associated with the packet transmission (Tx) and reception (Rx) events. Usually, there are errors δ between the time the timestamps are taken and the time the Tx/Rx events happen. Therefore, the actual relation of timestamps is:

$$T_1^P - T_2^S = \Delta - T_{\text{delay}}^{P\to S} + \delta_{\text{Tx}}^P - \delta_{\text{Rx}}^S \tag{3}$$

$$T_4^P - T_3^S = \Delta + T_{\text{delay}}^{S\to P} + \delta_{\text{Rx}}^P - \delta_{\text{Tx}}^S, \tag{4}$$

where $\delta_{\text{Rx/Tx}}^{P/S}$ denote the errors of Tx/Rx timestamps of the Primary/Secondary clock receptively. Equation (2) becomes:

$$\frac{(T_1^P - T_2^S) + (T_4^P - T_3^S)}{2} = \Delta - \frac{(\delta_{\text{Tx}}^P - \delta_{\text{Tx}}^S) + (\delta_{\text{Rx}}^P - \delta_{\text{Rx}}^S)}{2}. \tag{5}$$

As $\delta_{\text{Rx/Tx}}^{P/S}$ is unknown, the offset calculation by (2) contains errors and affects the synchronization accuracy.

In Ethernet hardware PTP implementations, timestamps are taken by the NIC hardware, and thus $\delta_{\text{Rx/Tx}}^{P/S}$ are relative stable and can be compensated. $|\Delta|$ calculated by (2) can be limited to sub-*ns*. Software PTP timestamps packets in software routines. The uncertainties of software stack bring about variances in $\delta_{\text{Rx/Tx}}^{P/S}$ and result in synchronization errors at the *ms*-level. In LAN situations, as hardware PTP has been widely supported by Ethernet NICs, software PTP is usually used for testing or a temporary choice when there is no compatible PTP NICs.

3 Measurement Methodology

3.1 Testbed

Hardware. The PCs are all Lenovo ThinkCentre M920t-D224 with identical hardware (Intel i5-8500 CPU, 4 GiB RAM). The motherboard has three PCIe slots. The high throughput x16 slot is directly connected to the CPU, which is normally used by PCIe GPU. The remaining $\times 1$ and $\times 4$ peripheral slots are connected to the CPU through a PCIe switch in the Intel chipset. As the peripheral PCIe slots and the GPU slot have different access latency profiles [32], the WNICs in our experiments are all plugged into the $\times 1$ peripheral slot. In fact, PCIe latency is closely related to PTP performance, but it normally does not affect the accuracy, see discussions in §4.2.1 and §5.2.1.

We use the Jetson Xavier NX development kit [14] (6 ARMv8.2 cores @1.4GHz, 8 GiB RAM, by default, 4 cores are disabled) to evaluate the performance of mobile platforms. The Jetson board by Nvidia has a powerful GPU, which is irrelevant to synchronization, but this board is popular for prototyping video-based edge applications, hence we adopt it for the study. The board has two PCIe slots. One is used for WNIC and the other for Ethernet NIC through an M.2. to PCIe adapter [4].

We use Atheros AR9388, an ath9k-compatible WNIC, for the measurement. ath9k is a mature open source WNIC driver that we can modify and work on. In addition to WNICs, the PCs and mobile platforms are equipped with Intel I210 Ethernet NICs, which support hardware PTP and are mainly used for measuring the accuracy of Wi-Fi synchronization (see §3.3 and Figure 3).

We mainly use the above hardware platforms to present the measurement results, but our findings and conclusions are verified with several similar platforms: Hikey 970 development board [9] (8 ARM cores), ODYSSEY board [16] (Intel Celeron J4105 CPU), and ThinkCentre (Intel i5-6500 CPU).

Software. The operating system running on PCs is Debian GNU/Linux 9.8 (stretch) with kernel 4.19.37. The Jetson board uses Debian with kernel 4.9.6 maintained by the community. NONE of them are the real-time (rt) version. We use `hostapd` to set up Wi-Fi Access Points on PCs. `linuxptp` [15] is a user space daemon that implements the PTP protocol, and sends and receives PTP packets through the network socket. It is also responsible for synchronizing two clocks in a local system through the PTP kernel interface [17], *i.e.*, sync the system clock to the hardware PTP clock or vice versa. We do not modify `linuxptp` nor the PTP kernel interface. Instead, our implementation patches the ath9k driver to allow COTS Wi-Fi NICs to work properly with the existing PTP software stack.

3.2 Measurement Settings

When applying PTP to sync Wi-Fi connected devices belonging to different APs, Wi-Fi APs might play different roles. When they are in the transparent clock mode [1], Wi-Fi clients are directly synchronized to the primary clock, which is hosted, for example, by a computer connected to the backbone Ethernet. APs only help to forward PTP packets from/to the primary clock. However, they need to compensate for the forwarding delay, which is hard to achieve in COTS APs. To avoid such complexities, we adopt the boundary clock

[1]Assuming identical PHY rate or the transmission time is compensated.

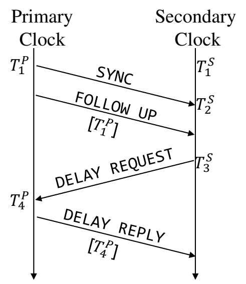
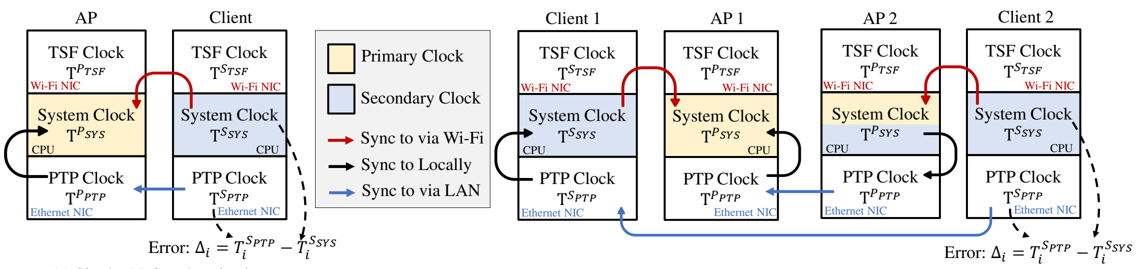


Figure 2: PTP Protocol.



(a) Single-AP Synchronization

(b) Cross-AP Synchronization

Figure 3: Measurement Method.

mode [1], where the synchronization chain is divided into multiple domains. Wi-Fi clients are first synchronized to their primary clocks hosted by their APs. The APs are then synchronized to a primary clock connected to the LAN backbone. In this way, all of the clients, whether within the same AP or not, are synchronized.

As the LAN PTP is mature, we assume the APs are correctly synchronized. The boundary clock mode allows us to focus on the single-hop wireless synchronization within one AP. The following measurements are firstly conducted in the Single-AP situation to study the core impacting factors, and then in the Cross-AP situation to show the feasibility.

3.2.1 Single-AP Situation

The measurement settings are shown in Figure 3 (a). A PC is used as the Wi-Fi AP. It also hosts the primary clock. The associated Wi-Fi clients are either PCs or Jetson boards. The synchronization frequency (each round of procedures in Figure 2 counts as one time) is 4 times per second. For experiments other than long-term tests, synchronization data is collected for 10 mins, *i.e.*, about 2400 offset samples (the first half is dropped to wait for synchronization algorithms to converge [39]). Experiments are conducted in a typical lab environment, using the IEEE 802.11a protocol (5 GHz, single antenna) in a relatively clean channel. To evaluate the performance, we traverse the following factors:

Network and System Factors are the network protocol and system-specific factors that the end systems can control and affect.

• CPU load. We study the impact of computational resources of the host system by varying the CPU usage to 0%, 50%, and 100%. `stress-ng` is used to generate CPU workload at the secondary host (client) by continually performing floating point calculation. We keep the CPU load of the primary host (the AP) unchanged, as asymmetrical conditions are more likely to introduce synchronization errors. CPU turbo boost is disabled and WNIC driver is bound to a fixed core to avoid the interference on the measurement.

• Network traffic. A socket program is developed to generate UDP traffic between the AP and clients. (a) The intensity of the traffic is characterized by the packet rate, *i.e.*, packet per second (PPS). The fine-grained traffic pattern is randomized by varying the inter-packet space. (b) The size of the packet is the number of bytes in the UDP payload. (c) We use `iwconfig` to fix the PHY rate from 6 Mbps to 54 Mbps.

Note that different combinations of PPS, packet size, and PHY rate result in different amounts of *channel idle time* and *throughput*, which, according to our evaluation, have no obvious impact on the results, hence we do not list them as independent factors. Further, the network traffic is bidirectional, but we find the impact of the uplink traffic (from client to AP) is less obvious, thus only the downlink cases are presented in the paper, *i.e.*, both CPU and network load are applied to the Wi-Fi clients.

Wireless-specific Factors. Wireless networks differ dramatically from wired networks in channel properties. We consider mobility, signal strength, and neighboring interference. These factors are out of the control of the end systems and have obvious impacts on general wireless transmissions. The mobility is introduced by moving and rotating the client. The received signal strength is controlled by placing the client to different locations. The interference is generated by a signal jammer, which is another pair of PCs sending random packets with Wi-Fi carrier sense turned off.

3.2.2 Cross-AP Situation

Based on the Single-AP setup, we measure Cross-AP synchronization by syncing APs via the LAN PTP. Figure 3 (b) is the measurement setup. Client 1 and Client 2 are synchronized by first syncing Client 1/2 to AP 1/2 and then syncing AP 2 to AP 1. The two APs are directly connected by Ethernet. A scalable way to sync more APs multiple hops away is using PTP switches. Note that the system clock of AP 2 is a boundary clock serving as the primary clock for its clients, and it is also a secondary clock from the point of view of AP 1 that AP 2 is synchronized to. We denote the synchronization chain with arrows: Client 1 $\xrightarrow{\text{sync to}}$ AP 1 $\xleftarrow{\text{sync to}}$ AP 2 $\xleftarrow{\text{sync to}}$ Client 2. In cross-AP measurements, the CPU and network load are only applied to Client 1. APs and Client 2 are load free.

3.3 Performance Metrics

In the Single-AP situation in Figure 3 (a), $T^{S_{\text{SYS}}}$ is synchronized to $T^{P_{\text{SYS}}}$ via Wi-Fi. To measure its accuracy, we use LAN PTP as the reference. The AP and client hosts are connected by an Ethernet link. The PTP clock of the client's Ethernet NIC is synchronized to AP's system clock through hardware LAN PTP. Note that the difference between $T^{S_{\text{PTP}}}$ and $T^{P_{\text{SYS}}}$ is at the ns-level, thus we use $\Delta_i = T_i^{S_{\text{PTP}}} - T_i^{S_{\text{SYS}}}$ to measure the synchronization error. The mean and standard deviation (std) of Δ_i, indicating bias and stability (jitter) respectively, characterize the synchronization performance. In the Cross-AP situation, we care about how clients belonging to different APs are synchronized, and thus use the Ethernet link to connect the two clients for comparing the difference of their system clocks through the LAN PTP. The method is similar to the Single-AP situation and shown in Figure 3 (b).

4 Software PTP

This section first describes our implementation of software PTP (§4.1). We analyze its performance from a system perspective (§4.2.1). Based on that, we propose system tuning techniques to improve the performance (§4.2.2) and a calibration procedure for using with diverse host platforms (§4.2.3). We conclude the trade-offs and limitations in §4.3.

4.1 Implementation - Software PTP

The PTP mechanism does not discriminate between Wi-Fi and Ethernet, but software PTP implementations do require NIC driver support to achieve reasonable accuracy. A user space program can take timestamps on its network socket but this just leads to a trivial NTP implementation, which has accuracy at the level of tens of *ms* [37]. According to equation (3)(4), software timestamps should be taken as close as possible to the Rx/Tx events to avoid software uncertainties. Therefore, Ethernet software PTP implementations normally take timestamps through the assistance of the NIC driver. However, we have not noticed any WNIC drivers provide such support, so we implement one for ath9k.

As WNICs use interrupts to inform the operating system about the Rx/Tx events, the system clock of the host is used to timestamp the PTP Tx/Rx events in the WNIC's interrupt service routine (ISR). The timestamps are first filled into a PTP-reserved field of packets' socket buffer, and then conveyed to and used by the kernel PTP subsystem and userspace PTP applications, *e.g.*, `linuxPTP`.

4.2 Findings and Analysis - Software PTP

Through measuring our software PTP implementation, the preliminary finding is that its synchronization is neither accurate nor stable. For example, when syncing two PCs under 48 Mbps PHY, 64 Byte size, 6000 PPS background traffic, the mean error is -1134 μs with std 992.5 μs. Results of using Jetson as the client are even worse, the mean error is -1892 μs with std 2947 μs. The performance is not consistent with the existing studies having similar implementations but with older platforms [33]. Our newer platforms perform much worse. To uncover the reasons, we look into the cause of the inaccuracy.

4.2.1 Root Cause of Inaccuracy

PTP accuracy is determined by T_{delay} and $\delta^{P/S}_{\text{Rx/Tx}}$ in equation (3)(4). For transmissions between Wi-Fi clients and AP, $T^{P\to S}_{\text{delay}} = T^{S\to P}_{\text{delay}}$, so the inaccuracy is caused by $\delta^{P/S}_{\text{Rx/Tx}}$. To characterize $\delta^{P/S}_{\text{Rx/Tx}}$, we measure its value of our platforms.

As shown in Figure 4, the measurement is aided by the TSF counter. When a packet is received, a WNIC takes several actions. First, it records the value of its TSF counter as $n^{\text{TSF}}_{\text{Rx}}$ at the time when the last bit of the packet is decoded. Second, the WNIC generates an interrupt to notify the operating system and also provides a brief description of the received packet, which contains $n^{\text{TSF}}_{\text{Rx}}$. Third, in the ISR of the WNIC driver, the operating system handles the Rx interrupt and takes software timestamp T^{SYS}. Once the software timestamp is taken, we read and record the TSF counter as $\dot{n}^{\text{TSF}}_{\text{Rx}}$. Since the I/O latency of reading TSF counter is relatively stable and small, $\dot{n}^{\text{TSF}}_{\text{Rx}} - n^{\text{TSF}}_{\text{Rx}}$ is just the number of 1 MHz ticks that $\delta_{\text{Rx}} + \frac{T^{\text{TSF}}_{+} - T^{\text{TSF}}}{2}$ takes [2]. δ_{Rx} can then be obtained. As when a packet is transmitted, the WNIC also reports the Tx status through an interrupt, the process of measuring δ_{Tx} is similar. Moreover, as δ_{Tx} is close to δ_{Rx} and their trend is identical, we use $\delta_{\text{Rx/Tx}}$ to describe them interchangeably.

The results of $\delta_{\text{Rx/Tx}}$ of software timestamps are shown in Table 1, from which we identify two system optimization mechanisms, Interrupt Mitigation/Throttling and CPU Power Management, which increase $\delta_{\text{Rx/Tx}}$ and cause large jitters, corrupting the synchronization performance.

Interrupt Mitigation. Interrupt mitigation is a mechanism to reduce CPU overhead by aggregating NIC interrupts [13]. When multiple NIC events occur in a period, only one interrupt is generated to notify the host system. The trade-off is, interrupts of most packets are delayed and the host system is less timely in response to network events. Interestingly, while this feature is well-known in Ethernet NICs, to our knowledge, it is rarely mentioned and evaluated under the COTS Wi-Fi context. Actually, we have not noticed any WNIC driver implements and provides the configuration interface, before our software PTP implementation.

When PPS is low (row 1 of Table 1), the mean and peak of $\delta_{\text{Rx/Tx}}$ are about 500 to 700 μs, indicating most packets need to wait for at least 500 μs for an aggregated interrupt. This is because, by default, the WNIC needs to wait for 500 μs to see if there is a consecutive packet following. When PPS increases, the inter packet space is less than 500 μs. The 500 μs threshold would lead to an unbounded waiting time, so the WNIC sets another threshold to limit the maximum waiting time, which is around 2000 μs (see peaks of row 3 of Table 1).

In cases with large asymmetric PPS, interrupt mitigation affects $\delta^P_{\text{Tx}} - \delta^S_{\text{Tx}}$ and $\delta^P_{\text{Rx}} - \delta^S_{\text{Rx}}$ in (5) by more than 2000 μs, causing the sync error at a similar level. Interrupt mitigation can be disabled by hacking the WNIC (see §4.3). After doing so, the distribution of $\delta_{\text{Rx/Tx}}$ becomes much sharper (see the peak and mean difference of row 4,5,6 of Table 1), but when the CPU load is light, its mean is still as large as 176 μs (row 4 of Table 1). The reason is the following factor.

[2] $T^{\text{SYS}}_{+} - T^{\text{SYS}}$ is the PCIe WNIC read latency. For our PC's $\times 1$ slot, it is about 1.7 μs. For Jetson's $\times 4$ slot, it is about 1.4 μs.

CPU C-state	Network Rx IMT	Network PPS	CPU Load 0%		CPU Load 50%		CPU Load 100%		
PC C0~C8	Enabled	100	674.3	699.2	600.3	519.2	519	518.2	1
		1000	558.8	557.2	546.8	519.2	520.1	518.2	2
		5000	1093	2046	1098	2042	1088	2010	3
	Disabled	100	175.6	203.2	105.3	23.24	20.5	20.24	4
		1000	60.31	59.24	42.36	22.24	20.15	20.24	5
		5000	37.53	21.24	23.92	21.24	19.95	20.24	6
PC C0 Only		100	19.92	20.24	19.89	20.24	20.16	20.24	7
		1000	19.75	20.24	19.93	20.24	19.91	20.24	8
		5000	19.66	19.24	19.67	19.24	19.93	20.24	9
Jetson C0 Only		100	16.75	15.58	16.24	15.58	16.5	15.58	10
		1000	16.51	15.58	16.17	14.58	16.54	15.58	11
		5000	15.89	14.58	16.61	15.58	16.65	15.58	12
			A	B	C	D	E	F	

🟦 Mean 🟩 Peak

Table 1: Error of Software Timestamp. Mean error and peak (mode) of the error distribution in unit of μs. The width of the color bars in each table cell illustrates the relative size of the underlying values. The bar of mean (blue)/peak (green) logarithmically grows from right to left/from left to right. Rx IMT is short for Rx Interrupt Mitigation.

CPU Idle Power Management. Modern CPUs leverage several mechanisms to dynamically conserve power [5]. When the CPU is idle, it might choose to turn off different numbers of hardware components, represented by different CPU idle states or C-states [43], *e.g.*, from C0 to C8 in Intel i5 8500. CPU in deeper C-state consumes less power, but takes more time to wake up to the active state C0, *e.g.*, to handle interrupts. As a result, power management mechanisms tend to let CPUs stay in lighter C-states when there likely are more intensive tasks or more frequent interrupts. This explains why $\delta_{Rx/Tx}$ decreases when CPU load and PPS increase (row 4,5,6 of Table 1). Different CPU and PPS loads lead to 150 μs difference in $\delta_{Rx/Tx}$, causing 150 μs mean error in synchronization. CPUs can be forced to stay in C0 state via its `sysfs`, in which it takes the least latency to respond to interrupts.

4.2.2 Software PTP with Proper System Configurations

Guided by the above analysis, we improve our software PTP implementation by disabling interrupt mitigation and forcing CPUs in C0 state. When the hardware of AP and client hosts is identical, the synchronization error in most cases is within 1 μs, with std less than 1 μs (A,C,E column of Table 2). However, it becomes less accurate when the hosts hardware is heterogeneous (the right subtable of Table 2). The reasons are rooted in the following factors.

In column AA, CC, and EE of Table 2, there is a constant bias of 4.x μs. This bias can be explained by comparing row 7-9 with row 10-12 of Table 1. Compared with Jetson, PC's software timestamps are delayed by 4.x μs, causing 4.x μs residues in $\delta^P_{Tx} - \delta^S_{Tx}$ and $\delta^P_{Rx} - \delta^S_{Rx}$ in (5), which leads to the bias error. What causes the difference between $\delta^P_{Rx/Tx}$ and $\delta^S_{Rx/Tx}$? Is ARM handling WLAN interrupts more timely than x86? These questions are out of our scope, but we have the following experience. Note that as both PC and Jetson are in C0 state in row 7-12 of Table 1, $\delta_{Rx/Tx}$ does not contain too much latency to wake up the CPU. Hence, $\delta_{Rx/Tx}$ mainly consists of two parts. One is attributed to the *WNIC interrupt latency*.

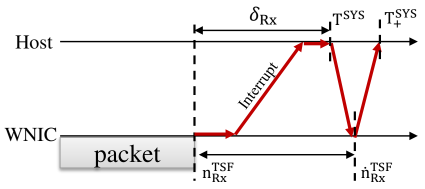


Figure 4: Using TSF Counter to Measure the Error of Software Timestamp.

There is an undocumented but stable delay between the end of Rx/Txing a packet and the issuing of the interrupt. This part is slightly different for the Rx and Tx chain, and different WNIC models. The remaining part (5 to 10 μs) characterizes the *responsiveness of the host system* in handling PCIe interrupts. Through extensive tests, we find this part varies with different host hardware, kernel versions, PCIe slots, and even CPU loads. Asymmetric interrupt responsiveness is common in host systems, which affects the PTP accuracy at the level of (less than) 10 μs.

4.2.3 Software PTP with Calibration

We discuss the calibration of the two types of bias errors.

Asymmetric Host Interrupt Responsiveness. Our insight is from the measurement method of Table 1. We note that the interrupt responsiveness is relatively stable when CPUs are in C0 and interrupt mitigation is off. Under such configurations, both the AP and client hosts can measure $\delta_{Rx/Tx}$ online when there is WLAN traffic, thus it is practically convenient. When using identical WNIC models, the *WNIC interrupt latencies* cancel out in (5), hence the secondary clock can subtract $\delta^P_{Tx} - \delta^S_{Tx}$ and $\delta^P_{Rx} - \delta^S_{Rx}$ from (5) to compensate for the bias of the host interrupt responsiveness.

Specifically, we measure all rows from 7 to 12 of Table 1. As the values are very close, only a subset is also enough. Note that the distribution of $\delta_{Rx/Tx}$ is single side and positive. The peak or mode of the measured $\delta_{Rx/Tx}$ is chosen as the specific value of $\delta_{Rx/Tx}$ for calibration (the mean value is affected by the long tail). We use the average of the peak values in Table 1 to compensate for the bias in Table 2. The compensated results are shown in Table 3. The mean error is less than 1 μs, which is comparable to the case using identical hardware (left subtable of Table 2). The std is $\times 3$ to $\times 5$ larger. The reason is the Jetson (ARM) platform is not as stable as the PC platform in handling interrupts. Jetson's $\delta_{Rx/Tx}$ contains a longer tail, which also can be observed from Table 1. *i.e.*, PC's mean and peak is closer than Jetson's.

To investigate the long-term stability, we log the synchronization error (after calibration) for 14 hours for the same hardware settings as Table 3. A script is used to generate random CPU and network load to the client during the measurement to emulate practical scenarios. The overall mean error is 0.17 μs with std around 1.8 μs, indicating stable and accurate synchronization between the two hosts. Due to space limitations, we omit the measurement results of cross-AP and wireless-specific factors for software PTP. As the findings and conclusions are identical to the hardware PTP, the reader can

Network			PC (Client) $\xrightarrow{sync\ to}$ PC (AP)							Jetson (Client) $\xrightarrow{sync\ to}$ PC (AP)						
			CPU Load							CPU Load						
Rate	Size	PPS	0%		50%		100%			0%		50%		100%		
/	/	0	0.11	0.38	-0.13	0.51	-0.19	0.48	1	4.69	1.73	4.2	1.72	4.65	1.61	1
6M	500B	100	0.13	0.37	0.17	0.5	0.25	0.54	2	4.39	1.69	4.93	1.7	4.69	1.52	2
		1000	-0.07	0.56	0.1	0.32	0.27	0.34	3	4.52	1.8	4.26	1.49	4.63	1.78	3
48M			0.54	0.43	-0.26	0.38	0.11	0.4	4	4.47	1.76	4.57	1.76	4.5	1.62	4
	64B		0.21	0.47	0.27	0.35	0.09	0.39	5	4.24	1.71	4.26	1.67	4.71	1.53	5
		5000	0.38	0.51	0.41	0.33	-0.06	0.35	6	4.46	1.51	4.58	1.81	4.54	1.56	6
			A	B	C	D	E	F		AA	BB	CC	DD	EE	FF	

mean 🟦

std 🟥

Table 2: Single-AP Software PTP Results. Rate: PHY rate of the Wi-Fi link (Mbps). Size: UDP payload length (Byte). PPS: packet per second. Values in the table are mean and standard deviation (std) of the synchronization error in unit of μs. The width of the color bars in each table cell illustrates the relative size of the underlying values. The bar of mean (blue)/std(red) linearly grows from right to left/from left to right. Negative mean values are shown in gray.

Network			Jetson (Client) $\xrightarrow{sync\ to}$ PC (AP)						
			CPU Load						
Rate	Size	PPS	0%		50%		100%		
/	/	0	0.20	1.73	-0.29	1.72	0.15	1.61	1
6M	500B	100	-0.10	1.69	0.43	1.7	0.20	1.52	2
		1000	0.03	1.8	-0.23	1.49	0.14	1.78	3
48M			-0.03	1.76	0.07	1.76	0.01	1.62	4
	64B		-0.26	1.71	-0.23	1.67	0.21	1.53	5
		5000	-0.04	1.51	0.09	1.81	0.04	1.56	6
			AA	BB	CC	DD	EE	FF	

Table 3: Single-AP Software PTP Results with Calibration. Notations are same as Table 2. This table is calculated from the right subtable of Table 2, taking the interrupt responsiveness of different platforms into account.

refer to §5.2.2 and §5.2.3.

The above calibration procedure is based on measuring $\delta_{Tx/Rx}$, which is the time between the WNIC taking TSF timestamp ($n^{TSF}_{Tx/Rx}$) and the host system taking software timestamp (T^{SYS}). When we treat $\delta_{Tx/Rx}$ as the error of software timestamp, there is an implicit assumption that the TSF timestamp $n^{TSF}_{Tx/Rx}$ and the packet event have a known and fixed association, *e.g.*, it is taken at the end of the packet, which, however, is not be true for different models of WNICs. Some models take TSF timestamps shortly after the end of the receiving, while others might take at the middle. We term this model-dependent factor as *TSF offset*. The *TSF offset* combined with the *WNIC interrupt latency* contributes about 10 μs to latencies in Table 1. While the *TSF offset* is unknown, it is fixed for the same model, hence it also automatically cancels out in equation (5) when using WNICs of the same model. However, when using different WNIC models, it needs offline calibration. We will revisit this issue later in §5.2.1.

WNIC Interrupt Latency is another hardware-related bias source. We note that the *TSF offset* only affects the above calibration, but *WNIC Interrupt Latency* is intrinsic in software PTP. When interrupt mitigation is disabled, the interrupt should be issued once the Tx/Rx event is finished, but in practice, there is a model-dependent delay, which can be observed when using different WNIC models on identical host platforms. Due to the coupling with the *TSF offset*, its value cannot be measured with the method in Figure 4. As a model-dependent factor, it needs offline calibration similar to *TSF offset*.

4.3 Discussion - Software PTP

Software PTP timestamps packets in ISR, which is a standard software routine in the network stack. Under certain configurations, software PTP can achieve inspiring results. The accuracy is μs-level and the std is within 2 μs, which is accurate enough for many time-sensitive applications. Next, we discuss the trade-offs and limitations.

Power Efficiency. Interrupt mitigation and CPU idle states are critical for the software PTP performance. However, completely disabling them is a concern for most mobile devices, as they are limited in battery life. A general workaround is duty cycling. Since PTP does not need to frequently exchange packets, PTP packets can be pre-scheduled at given time slots, during which the mobile host switches to the "active" mode (with C0 and IMT off) for precisely timestamping PTP packets. Additionally, there are other specific alternatives. For example for retaining idle states, the AP can choose to preheat the client by sending a dummy packet before every PTP packet. The client will be wakened up by the pre-heat packet. For reducing the number of interrupts while keeping responsive to PTP packets, PTP traffic can be assigned to the Rx priority queue (see the following paragraph). Moreover, since the PTP measurement is periodical, it has a negative impact on the IEEE 802.11 power-saving (PS) mode. We will discuss this issue later with hardware PTP in §5.3.

Hardware Compatibility. While software PTP is mainly finished in software routines, it depends on disabling interrupt mitigation to improve the PTP performance, which is not a typical configurable parameter in COTS WNICs. We achieve this in two ways. The first is to completely disable this feature through specific registers, which might lead to too many interrupts overwhelming the host system. Another way is to prioritize PTP packets. By default, Tx interrupt mitigation is disabled by the driver. The WNIC we tested implements two hardware Rx queues with different priorities. Packets with high IEEE 802.11 QoS priority are handled by the high-priority Rx queue, and their interrupts are delivered instantly without the mitigation delay. The above two approaches are all hardware-dependent features. While we have not confirmed the compatibility of other WNICs except for the ath9k series, we think such features are functionally reasonable and are likely supported by other modern WNICs.

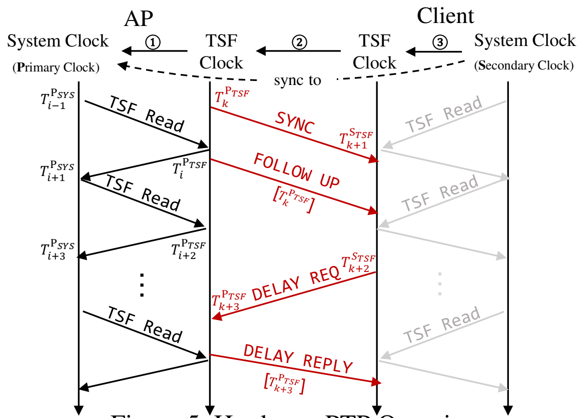


Figure 5: Hardware PTP Overview

5 Hardware PTP

This section first describes our implementation of hardware PTP for Wi-Fi devices (§5.1). Then we analyze its performance by considering the host system and hardware factors (§5.2.1). As a wireless synchronization approach, the impact of mobility and wireless channel conditions are considered in §5.2.2. Additionally, §5.2.3 demonstrates the feasibility of cross-AP synchronization. We summarize the benefit and limitations of hardware PTP in §5.3.

5.1 Implementation - Hardware PTP

Hardware PTP needs hardware counters to timestamp network packets, but how can the WNICs support a hardware feature that they are not originally designed for?

Our key insight is to treat the TSF clock as the hardware timestamping clock. The feasibility is based on two features. First, WNIC uses the TSF counter to timestamp network packets, for the purpose of protocol debugging, packet sniffing, *etc.* Second, the TSF counter can be accessed by the host operating system, through reading the TSF counter registers. The above features are the necessary conditions for a PTP hardware clock. By using the TSF counter, hardware PTP can be realized in Wi-Fi NICs in the same way as the Ethernet NICs. This subsection highlights the major procedures (⓪-③) shown in Figure 5:

⓪ Generate $T^{P_{\text{TSF}}}$ and $T^{S_{\text{TSF}}}$. According to equation (1), we maintain a TSF clock for each WNIC based on its TSF counter. The time of the clock is denoted as T^{TSF}, which is determined by TSF counter and the clock parameters $\{T_{\text{off}}^{\text{TSF}}, R^{\text{TSF}}\}$.

① Sync $T^{P_{\text{TSF}}}$ to $T^{P_{\text{SYS}}}$. The host of the primary clock, *i.e.*, the AP, synchronizes its TSF clock to its system clock. The method is to read each clock and tune TSF clock's parameters $\{T_{\text{off}}^{P_{\text{TSF}}}, R^{P_{\text{TSF}}}\}$ to minimize the difference between $T^{P_{\text{SYS}}}$ and $T^{P_{\text{TSF}}}$. In practice, the latency and jitter of obtaining timestamps must be taken into consideration. We adopt a sandwich-like reading sequence to access T^{SYS} and T^{TSF} multiple times, and only the one with the smallest read latency $(T_{i+1}^{P_{\text{SYS}}} - T_{i-1}^{P_{\text{SYS}}})$, say k, are used to estimate their clock offset, *i.e.*, $\Delta = T_k^{P_{\text{TSF}}} - (\frac{T_{k-1}^{P_{\text{SYS}}} + T_{k+1}^{P_{\text{SYS}}}}{2})$.

② Sync $T^{S_{\text{TSF}}}$ to $T^{P_{\text{TSF}}}$. The host of secondary clock synchronizes its TSF clock to $T^{S_{\text{TSF}}}$ according to PTP protocol in Figure 2. Note that the WNIC hardware automatically timestamps packets with TSF counter. The value of the counter is transformed to TSF time according to ⓪. After each round of measurement, the time offset of two TSF clocks is estimated by $((T_1^{P_{\text{TSF}}} - T_2^{S_{\text{TSF}}}) + (T_4^{P_{\text{TSF}}} - T_3^{S_{\text{TSF}}}))/2$, according to which the host of $T^{S_{\text{TSF}}}$ tunes its TSF clock's parameters $\{T_{\text{off}}^{S_{\text{TSF}}}, R^{S_{\text{TSF}}}\}$ to minimize the offset between $T^{S_{\text{TSF}}}$ and $T^{P_{\text{TSF}}}$. Since $T^{P_{\text{TSF}}}$ has been synchronized to the primary clock $T^{P_{\text{SYS}}}$ in ①, $T^{S_{\text{TSF}}}$ is thus synchronized to $T^{P_{\text{SYS}}}$.

③ Sync $T^{S_{\text{SYS}}}$ to $T^{S_{\text{TSF}}}$. The host of secondary clock synchronizes $T^{S_{\text{SYS}}}$ to $T^{S_{\text{TSF}}}$ by tuning parameters $\{T_{\text{off}}^{S_{\text{SYS}}}, R^{S_{\text{SYS}}}\}$ like step ①. As $T^{S_{\text{TSF}}}$ has been synchronized to $T^{P_{\text{SYS}}}$ in step ②, the system clocks of two hosts, *i.e.*, $T^{P_{\text{SYS}}}$ and $T^{S_{\text{SYS}}}$, are then synchronized.

The above procedures mimic the implementation of hardware PTP of Ethernet NICs. However, unlike the Ethernet case, the TSF counter is not a free-running counter in Wi-Fi NICs. We analyze the impact of this difference in the following subsection.

5.2 Findings and Analysis - Hardware PTP

The measurement experiments are identical to the software PTP situation. The results are shown in Table 4. In all the cases, the mean error is less than 1 μs with std less than 1 μs. Compared with software PTP, no obvious difference is observed when using different host platforms. This is because the timestamps are taken by the WNIC hardware, and involve no operations of the host system. A similar long-term measurement is conducted using Jetson as the client. The overall mean error is -0.16 μs with std around 0.5 μs. The sync std/jitter is smaller than the software PTP.

5.2.1 Potential Cause of Inaccuracy

While the accuracy is close to the resolution of the TSF counter (1 MHz), we investigate the following factors to provide a sound understanding on why and under what conditions the hardware PTP implementation works well.

Symmetry of PCIe Single-way Latency. In step ① and ③ in §5.1 and Figure 5. The TSF clock is obtained by accessing the TSF counter. This procedure relies on a PCIe read action to the WNIC registers, *i.e.*, memory-mapped I/O (MMIO) read. To account for the read latency, the read action is sandwiched by two timestamps of the system clock and the average of the two system timestamps is used to approximate the actual time of accessing the TSF counter. In fact, the read action contains a read request and a read reply, *i.e.*, a round trip. As a consequence, the estimation method assumes that the latencies of the read request and reply are equal, *i.e.*, the PCIe single-way latency should be symmetrical.

We are not able to quantitatively validate the symmetry due to the lack of a PCIe analyzer. However, in general, it should approximately hold, as the same problem exists in Ethernet

Network			PC (Client) $\xrightarrow{sync\ to}$ PC (AP)							Jetson (Client) $\xrightarrow{sync\ to}$ PC (AP)						
			CPU Load							CPU Load						
Rate	Size	PPS	0%		50%		100%			0%		50%		100%		
/	/	0	0.18	0.37	0.2	0.44	-0.17	0.42	1	-0.23	0.53	-0.21	0.57	0.37	0.51	1
6M	500B	100	0.14	0.41	-0.03	0.37	-0.55	0.39	2	0.03	0.62	-0.07	0.51	-0.03	0.52	2
		1000	0.34	0.42	-0.33	0.43	0.49	0.45	3	0.13	0.54	0.29	0.51	0.21	0.67	3
48M			-0.17	0.45	0.16	0.36	0.41	0.44	4	0.13	0.47	-0.07	0.6	-0.03	0.47	4
	64B		0.24	0.34	-0.22	0.42	0.16	0.38	5	0.27	0.4	-0.35	0.42	0.05	0.4	5
		5000	0.11	0.39	0.05	0.37	-0.13	0.43	6	-0.09	0.76	-0.1	0.6	-0.49	0.7	6
			A	B	C	D	E	F		AA	BB	CC	DD	EE	FF	

mean 🟦

std 🟥

Table 4: Single-AP Hardware PTP Results (μs). Notations are same as Table 2.

Config		PC (ROLE) $\xrightarrow{sync\ to}$ PC (AP)						
		CPU Load						
ROLE	BI	0%		50%		100%		
Client	1000	0.19	0.81	0.13	0.79	-0.15	0.77	1
AP		0.21	0.42	0.08	0.44	-0.13	0.45	2
	100	0.17	0.39	-0.15	0.45	0.03	0.38	3
		A	B	C	D	E	F	

Table 5: Hardware PTP with Free-running TSF Counter (μs). Notations are same as Table 2. ROLE is the Wi-Fi mode of host (either Client or Access Point). The TSF counter of client mode is not free-running and used for comparison. BI is short for the Beacon Interval in unit of Time Unit (TU, 1024 μs in IEEE 802.11). The default BI is 100 TU, *i.e.*, 102.4 ms.

NICs (according to results in Table 4, the impact of this factor is not obvious in our WNICs). During the test, we also notice that the read latency slightly varies with the platform load, this is again related to the power management mechanisms, but it does not break the symmetry.

We note that the read latency of different hosts or PCIe slots is usually different (recall footnote 2 in §4.2.1), but the difference does not affect hardware PTP. This is because PCIe read actions only happen locally on the host system to relate the system clock to the hardware NIC clock. Whether $(T_{k-1}^{P_{\text{SYS}}} + T_{k+1}^{P_{\text{SYS}}})/2$ can accurately estimate $T_k^{P_{\text{TSF}}}$ or not only depends on the symmetry of single-way latencies, rather than the total read latency $T_{k+1}^{P_{\text{SYS}}} - T_{k-1}^{P_{\text{SYS}}}$.

Impact of TSF Synchronization. Note that in PTP, the timestamping clock should be a free-running clock. This is only true for the Wi-Fi AP. The TSF counters of the Wi-Fi clients are automatically set to the value of the AP's TSF counter contained in the beacon packet broadcast by the AP every 102.4 ms. As many Wi-Fi timing functions, such as power save wake up, transmission scheduling, *etc.*, rely on TSF, therefore disabling the TSF synchronization would lead to a mess in the Wi-Fi network. Therefore, we design a new experiment to compare the performance of hardware PTP using/not using free-running TSF counters.

We use two PCs as *AP*s and set their system clocks as the PTP primary clock and secondary clock respectively. The two APs work in the same channel, which allows them to overhear the broadcast packets from each other. The unicase addresses of PTP packets are modified to the broadcast address, so that two APs can exchange PTP packets to allow hardware PTP to operate in the same way as before. Since they are all in the AP mode, their WNICs' TSF counters are free-running. The synchronization results are shown in Table 5.

Comparing row 3 of Table 5 and row 1 of Table 4, there is no obvious performance increase or reduction when using a free-running TSF counter. However, when the beacon interval is increased, *i.e.*, the TSF synchronization frequency is reduced. The case using the free-running counter stays the same (row 2 of Table 5), while the normal configuration becomes more unstable (row 1 of Table 5). This is because the TSF counter of the Wi-Fi AP is free-running, but the TSF counter of the Wi-Fi client is periodically set to beacons. Due to the frequency drift between the two TSF counters, the value of the client's counter jumps once it is reset by the beacons. The longer the client's TSF counter does not sync to the beacon, the larger the jumps are. Therefore, lower sync frequency results in larger jitters in the client's PTP clock $T^{S_{\text{TSF}}}$, and hence the sync results.

Comparing row 1 of Table 5 (BI=1000) and row 1 of Table 4 (BI=100), when the frequency of beacons is as high as the default (BI=100), the impact of TSF synchronization is limited due to two reasons. First, by default, the TSF counter is actually so frequently updated, that the jump values of the TSF counter are very small. For our WNICs, the clock drift is 20 ppm, meaning that the counter at most drifts 2 μs in each 100 TU. Second, the jumps of the TSF counter do not affect the TSF clock at the same level. This is because when an offset of TSF clocks is observed by the PTP protocol (②) of §5.1), $T^{S_{\text{TSF}}}$ is adjusted by the rate part, *i.e.*, parameter R in equation (1), and several control filters are used to smooth the impact of noise and outliers. Due to the above reasons, when the client's TSF counter is frequently updated to the AP's, it can be viewed as a "free-running" counter, which has exactly the same rate as the AP's TSF counter but with more jitters.

Impact of TSF Offset. Hardware PTP uses the TSF counter to timestamp packets, hence $\delta_{\text{Rx/Tx}}$ in (5) is free of the impact of the host system but is affected by the WNIC *TSF offset*. When WNICs have different *TSF offset*s, $\delta_{\text{Rx/Tx}}$ in equation (5) will not cancel out, which results in bias error. In another experiment not presented here, we replaced the client's WNIC with another model of Atheros NIC, then a constant positive 1 μs mean error occurred in all the cases. This implies $\delta_{\text{Rx/Tx}}$ does exist and differs across models. Other indirect evidence is from the driver source code. There are registers with magic numbers for compensating TSF timestamps, which directly affect the mean errors in Table 4. Similar to the software PTP, one can use WNICs of the same model to avoid *TSF offset* (the host platforms may differ), otherwise, an

Network			PC (Client 1) → PC (AP 1) ↖ PC (Client 2) → PC (AP 2) ↗						
			CPU Load						
Rate	Size	PPS	0%		50%		100%		
/	/	0	0.24	0.72	0.17	0.82	-0.11	0.85	1
6M	500B	100	-0.2	0.53	-0.2	0.69	0.32	0.75	2
		1000	-0.03	0.72	-0.18	0.72	-0.16	0.67	3
48M			0.12	0.6	0.18	0.87	0.19	0.67	4
	64B		0.18	0.67	0.2	0.78	0.24	0.56	5
		5000	0.21	0.55	0.01	0.63	0.11	0.58	6
			A	B	C	D	E	F	

PC (Client 1) → PC (AP 1) ↖ Jetson (Client 2) → PC (AP 2) ↗						
CPU Load						
0%		50%		100%		
0.14	0.96	0.12	1.07	0.02	1.19	1
0.43	1.04	0.52	0.82	0.8	0.74	2
0.38	0.85	-0.53	1.05	0.14	1.08	3
0.06	0.54	-0.06	0.62	0.04	0.59	4
-0.01	0.92	0.05	1.03	0.11	0.94	5
-0.05	0.96	-0.06	0.68	0.07	1.05	6
AA	BB	CC	DD	EE	FF	

mean	🟦
std	🟥

Table 6: Cross-AP Hardware PTP Results (μs). Notations are same as Table 2.

Factors	Acronyms	Meaning
Network & CPU Load	load ~~load~~	Network and CPU load are applied to (load) or not applied to (~~load~~) the client. When applying the load, we use the same configuration as cell 6-EE of Table 1, *i.e.*, 48 Mbps, 64 Byte, 5000 PPS, 100% CPU.
Mobility	static* walk rotate	The mobility is introduced through holding the client to walk or rotating it, while keeping the AP static. The moving speed introduced by walking and rotating is around 1 m/s and 2 m/s.
Signal Strength	-30 dBm -55 dBm* -75 dBm	The received signal strength is adjusted by placing the client in different locations. The value is measured and reported by the WNIC hardware on the per packet basis.
Interference	none* 100 PPS 250 PPS	Two hosts are configured to transmit Wi-Fi packets (64 Byte @ 6 Mbps) with carrier sense disabled. Their transmissions not only occupy the channel but can also interfere with on-going PTP transmissions.
	default	Default configurations for comparison are denoted with *

Table 7: Measurement Acronyms.

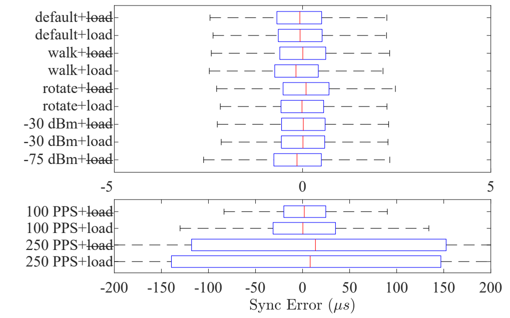


Figure 6: Sync Error Distribution v.s. Wireless Impacting Factors. The central mark indicates the median, and the left and right edges of the box indicate the 25th and 75th percentiles, respectively. Refer to Table 7 for the explanation of acronyms. There is no network load at -75 dBm case, as the SNR is not enough to decode 48 Mbps PHY rate packets.

offline calibration procedure is needed to measure and correct *TSF offset*. A typical solution (like the hardware offset in PTP Ethernet NICs) is to measure those model-dependent values with a reference clock, then the driver maintainer corrects them in the source code.

5.2.2 Wireless-Specific Impacting Factors

As a wireless synchronization approach, PTP over Wi-Fi may also be affected by wireless-specific factors. We study their impact by performing PTP under different mobility, signal strength, and interference levels. The error distributions are shown in Figure 6. In most cases, no obvious difference is observed from the default case, meaning that these factors do not affect PTP synchronization. This is because both hardware and software PTP depend on the accuracy of the timestamps of the PTP packets, but these wireless factors can not affect successfully-decoded packets[3]. Also, note that the neighboring interference significantly decreases the accuracy. This is because, due to the strong interference, PTP packets are jammed, postponed, and (mostly) dropped. As a result, the PTP algorithm does not have enough timestamps for clock adjustment, which significantly increases sync intervals and causes large jitters.

5.2.3 Cross-AP Synchronization

We conduct measurements of cross-AP synchronization (see Figure 3 (b)). Results are shown in Table 6. Compared with the Single-AP situation in Table 4, the mean error is similar. This is because when APs are synchronized boundary clocks, clients synced to one AP are also synced to other APs and clients. The synchronization performance among clients is determined by the wired link and the Wi-Fi link. The former is guaranteed by Ethernet PTP (ns-level), while the latter has been studied in our previous measurements (μs-level). However, jitters of the Cross-AP are more intense than that of the Single-AP. This is because Table 6 compares the clocks of client1 and client2 rather than clocks of client and AP. When two secondary clocks (hosted by client1 and client2) are synchronized to the same primary clock (hosted by AP1), their errors are independent random variables. The variance of their difference is the sum of their individual variances. This is a known limitation of the PTP boundary clock mode, whose error is accumulated along the sync chain [7].

5.3 Discussion - Hardware PTP

We discuss the potential improvement and power efficiency of hardware PTP, and then we compare it with software PTP.

Further Improvement. Hardware PTP syncs two hosts with high resolution system clocks (sub ns-level) via low-resolution (1 μs) TSF clocks. In principle, the synchronization resolution is related to the stability, rather than the resolution of TSF clocks. A finer resolution might be achieved by sending a burst of PTP measurement packets with a fixed inter-packet interval. It might be possible to infer or interpolate sub-μs timestamps by analyzing the rounding/stepping patterns of TSF timestamps.

Power Efficiency. Hardware PTP does not need to timely activate the host system for software timestamping, since

[3]Depending on the implementation, TSF timestamps might be related to the PHY-layer frame synchronization, which is affected by channel conditions. However, the error should be within one cyclic prefix, *i.e.*, 0.8 μs in 802.11a.

HOST	WNIC	Software PTP	Hardware PTP
same	same	✓	✓
diff.	same	int. calib.	✓
same	diff.	hw calib.	hw calib.
diff.	diff.	int.+hw calib.	hw calib.
timestamp packets		ISR	hw support
read TSF counter		no need	hw support
power efficiency		similar	

Table 8: Software and Hardware PTP Requirements

the timestamps have been taken by the WNIC hardware and stored in the meta info associated with the packets. However, PTP traffic has a negative impact on the IEEE 802.11 PS mode, where the WNIC of the client is aggressively turned down into a low power mode when there is no traffic. The AP buffers packets for the PS client and sends notifications through the beacons. The client WNIC wakes up at every beacon to check if there are packets for it, if there are, the WNIC will heuristically stay active for a certain period. As a result, the PTP traffic stimulates the WNIC periodically, making the PS mode deficient. We measure the impact with a power meter. When there is no traffic, the PS mode can save 0.37 W. When the PTP (4 PPS) is enabled, the PS can only save 0.15 W. Considering the overall standby power of the Jetson board is only 2 to 3 W, the trade-off between synchronization accuracy and power efficiency must be carefully considered when applying PTP to low-power applications.

On the other hand, when the PS is enabled, PTP packets will be delayed for 30 to 100 ms, but the delay does not affect the synchronization performance. This is because the timestamp of a buffered packet is taken when it is sent off to the air and when it is correctly received by the client.

Hardware v.s. Software PTP. From Table 2 and 4, both hardware and software PTP implementations can achieve μs-level accuracy with COTS Wi-Fi devices, which we believe is sufficient for most time sensitive mobile applications. Besides, they inherit the benefit of PTP and can extend the synchronization range across multiple APs. To make the choice between them, we summarize their requirements in Table 8. When using different hosts for the primary and secondary clocks, software PTP needs to calibrate the interrupt responsiveness (int.calib. see §4.2.3). When using different WNICs, they both need offline calibration to correct hardware offsets (hw calib. see §4.2.3 and §5.2.1). We note that to support hardware PTP, the WNIC should be able to timestamp packets with its TSF counter and also provide the counter reading interface (see §5.1). While TSF is a mandatory feature of IEEE 802.11, these features are not, hence they might not be available on COTS WNICs. In short, if there are supported WNICs, hardware PTP is a convenient and accurate choice for Wi-Fi synchronization. Otherwise, software PTP is a more general and flexible choice with different levels of accuracy requiring different levels of calibration.

6 Other Wi-Fi Synchronization Protocols

This section compares PTP with TSF and FTM.

TSF synchronization is automatically finished by the WNIC hardware. The AP broadcasts the value of its TSF counter in beacon frames, according to which its client WNICs reset their TSF counters. Since TSF is originally designed for timing functions of Wi-Fi protocol, the TSF counter by default is not related to the host system clock. Therefore, we measure the difference between the TSF counters of the AP and the client to estimate the synchronization error. Their TSF counters are read and sandwiched by synchronized Ethernet PTP clocks like Figure 5. The time offset between the client and AP counter is calculated and shown in Table 9.

Similar to the hardware PTP implementation, errors of TSF synchronization are independent of the host system and the system load. There is a bias error of -4 μs and the jitter is larger. The bias error is attributed to different TSF offsets of the Tx and Rx chain, *i.e.*, $\delta_{\text{Tx}} \neq \delta_{\text{Rx}}$. Bias is a common error source of single-way synchronization methods. The jitter is caused by the TSF counter jumps caused by the frequency drift. Additionally, TSF cannot be directly used to sync clocks in the cross-AP situation. We note that while the above problems are not fundamental, for example, the bias error can be calibrated, a complete solution would probably just lead to our hardware PTP implementation, which makes use of the two-way PTP measurement to factor the offset off and leverages existing PTP interface to sync clocks in a scalable manner.

FTM/TM is the latest feature of the IEEE 802.11 standard. They use dedicated WNIC hardware for timestamping. FTM and TM differ in resolution. TM is at the level of 10 ns, while FTM is ps-level. FTM's high time resolution is designed for measuring the time-of-flight (ToF) of radio waves for ranging. Another goal of TM/FTM (IEEE 802.1as Clause 12 [2]) is to extend the LAN PTP synchronization to Wi-Fi devices, targeting the same problem as our PTP implementations.

Although some COTS WNICs support FTM for localization [27], they rarely expose the synchronization interface. After extensive tries, the only feasible way we found (by referring to the PTP daemon `gptp` [8] managed by Intel) to enable synchronization of FTM WNICs is: 1. using Intel FTM WNICs (Intel 8260 AC) with 2. proprietary Windows driver [11] 3. in the Wi-Fi direct mode (Ad-Hoc mode). We also note that even though it does work, the running protocol is TM rather than FTM (they differ in time resolution), which is observed through packet sniffing.

Given the above hardware and software, we are still not able to measure the accuracy of TM/FTM as Windows 10 does not natively support hardware PTP and Windows Server does not support the WNIC driver, thus the ground truth cannot be obtained. However, the std of the sync error can be measured and is shown in Table 10. The std is around 0.4 μs, which is close to our hardware PTP based on the TSF counter, but

Network			$\text{PC (Client)} \xrightarrow{sync\ to} \text{PC (AP)}$							$\text{Jetson (Client)} \xrightarrow{sync\ to} \text{PC (AP)}$						
			CPU Load							CPU Load						
Rate	Size	PPS	0%		50%		100%			0%		50%		100%		
/	/	0	-3.78	1.09	-3.62	0.96	-3.67	1.13	1	-4.08	1.06	-3.83	0.93	-3.67	1.32	1
6M	500B	100	-3.67	1.16	-3.53	1	-3.65	0.96	2	-3.86	1.09	-3.49	1.16	-3.71	0.94	2
		1000	-3.91	0.99	-3.79	0.98	-3.71	0.99	3	-3.74	1.05	-3.83	1.18	-3.82	1.15	3
48M			-3.76	1.14	-3.88	0.85	-3.67	1.01	4	-3.99	1.15	-4.07	1.2	-3.78	0.99	4
	64B		-3.8	0.91	-3.72	0.98	-3.82	1.12	5	-3.92	0.92	-4	1.09	-3.85	1.04	5
		5000	-3.69	1.06	-3.68	1.04	-3.79	0.98	6	-3.87	1.07	-3.79	0.96	-3.71	1.12	6
			A	B	C	D	E	F		AA	BB	CC	DD	EE	FF	

mean 🟦

std 🟥

Table 9: TSF Synchronization Results. Values are in unit of μs. Notations are same as Table 2.

Network	CPU Load		
	0%	50%	100%
0%	0.39	0.44	0.41
50%	0.36	0.36	0.34
100%	0.37	0.38	0.39

std 🟥

Table 10: TM Results. Values are std of estimated synchronization error in unit of μs. CPU load and network load (represented by channel occupancy) are generated by `CPUSTRES` and `PSPING` from `Sysinternals` [19]

this value is actually very large compared with its 10 ns-level timestamp resolution. The reason is unknown but it is probably caused by the incomplete implementation or being intentionally disabled/hidden by the Intel firmware.

We believe FTM and TM provide an ideal solution for Wi-Fi synchronization, but it seems its primary goal is positioning and the synchronization feature is publicly unavailable right now. This is probably because FTM is only an optional feature of IEEE 802.11, or simply because of market reasons.

7 Related Work

Due to the importance of synchronized time in distributed systems [30], network synchronization protocols have been extensively discussed [42]. In a small network scale like LANs, the network delay between nodes is stable under prioritized switching [47] and the packets can be accurately timestamped by NIC hardware. These facts guarantee the performance of LAN PTP, and in practice it achieves nanosecond accuracy with commercial Ethernet NICs [25]. It is natural to extend PTP to wireless LANs due to the synchronization demand of mobile systems [44]. Corresponding practices started a decade ago and a comprehensive survey is given by [34]. We next review software and hardware PTP separately.

For hardware PTP, existing work focuses on improving timestamp accuracy and precision. The challenge is that wireless channels bring uncertainties to the received RF signals, making it hard to determine the accurate start of the packet. Modern OFDM-based and wideband PHYs allow the accuracy to reach several ps. However, most research work is based on laboratory prototypes [22, 24, 28] and commercial FTM WNICs are mainly for ranging and positioning purposes [27]. As a result, the performance of hardware PTP with COTS Wi-Fi devices remain unknown. Our work makes hardware PTP available on COTS devices and then characterizes its performance.

Our hardware PTP implementation is based on the TSF counter, so it is related to TSF-based synchronization approaches. RF-WiFi [45] makes use of TSF interrupts to notify the host systems so that they can take synced actions. Mahmood *et. al.* study the IEEE 802.11 Timing Advertisement feature [36], which uses beacon frames to convey AP's system clock to clients. However similar to TSF, it is single-way and thus has a bias. We have not encountered any works which make use a TSF counter as a hardware PTP clock.

For software PTP, the majority of existing work was conducted ten years ago with IEEE 802.11b Atheros WNICs. As a result, most insights no longer sufficient to light the current situation due to the rapid evolution of mobile systems. For example, the PCI DMA latency [33] becomes a minor issue since newer WNICs use PCIe [40]. The TSC clock is stable and independent of the core frequency [35] due to the adoption of invariant TSC in modern processors [21]. The impact of the CPU and network load is studied by existing work [26, 33], but our measurement reveals that they are not major factors.

Existing discussions on CPU idle power management focus on server platforms [43]. Our work reveals its impact on mobile platforms and on synchronization performance. Further, we have not noticed any discussion on interrupt mitigation under the context of Wi-Fi networks. As far we know, Atheros 802.11b WNICs does not have this feature, and hence past work based on old WNICs is able to achieve stable software PTP performance even without realizing this factor.

8 Conclusion

Accurate time synchronization is a key enabler for many distributed control and sensing applications. In this work, we study the performance of PTP protocol in Wi-Fi networks through systemic implementation and rigorous evaluation. The in-depth discussion and open-source implementation render a useful reference for designing and adopting PTP in Wi-Fi networks.

Acknowledgements

We thank anonymous reviewers for their valuable comments. We sincerely thank our shepherd Lin Zhong. We thank Shauna Dalton for the proofreading. This work is supported (in part) by the ShanghaiTech Startup Fund, the Shanghai Sailing Program 18YF1416700, the "Chen Guang" Program 17CG66 supported by Shanghai Education Development Foundation and Shanghai Municipal Education Commission, and NSFC 62002224.

References

[1] Ieee standard for a precision clock synchronization protocol for networked measurement and control systems. *IEEE Std 1588-2008 (Revision of IEEE Std 1588-2002)*, pages 1–300, July 2008.

[2] Ieee standard for local and metropolitan area networks - timing and synchronization for time-sensitive applications in bridged local area networks. *IEEE Std 802.1AS-2011*, pages 1–292, March 2011.

[3] Ieee standard for information technology—telecommunications and information exchange between systems local and metropolitan area networks—specific requirements - part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications. *IEEE Std 802.11-2016 (Revision of IEEE Std 802.11-2012)*, pages 1–3534, Dec 2016.

[4] Adt-link pci express 3.0 x4 to m.2 nvme extension cable. http://www.adt.link/product/R42U.html, 2021.

[5] Advanced configuration and power interface (acpi) specification, revision 6.3. https://uefi.org/sites/default/files/resources/ACPI_6_3_final_Jan30.pdf, May 2021.

[6] Ccnt. http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0211h/Bihcgfcf.html, 2021.

[7] Configuring precision time protocol (ptp). https://www.cisco.com/c/en/us/td/docs/switches/lan/catalyst9300/software/release/16-10/configuration_guide/lyr2/b_1610_lyr2_9300_cg/configuring_precision_time_protocol__ptp_.pdf, 2021.

[8] gptp. https://github.com/AVnu/gptp, 2021.

[9] Hikey970 evaluation board. https://www.96boards.org/product/hikey970/, 2021.

[10] Intel 64 and ia-32 architectures software developer manuals. http://www.intel.com/products/processor/manuals/, 2021.

[11] Intel proset/wireless software and drivers. https://downloadcenter.intel.com/download/30280, 2021.

[12] Intel true view. https://www.intel.com/content/www/us/en/sports/technology/true-view.html, 2021.

[13] Interrupt moderation of intel nics. https://www.intel.com/content/www/us/en/support/articles/000005811/network-and-io/ethernet-products.html, 2021.

[14] Jetson xavier nx module and developer kit. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-xavier-nx/, 2021.

[15] The linux ptp project. http://linuxptp.sourceforge.net, 2021.

[16] Odyssey x86 board. https://www.seeedstudio.com/ODYSSEY-X86J4105800-p-4445.html, 2021.

[17] Ptp hardware clock infrastructure for linux. https://www.kernel.org/doc/html/latest/driver-api/ptp.html, 2021.

[18] Wi-ptp. https://github.com/Wireless-Lab/Wi-PTP, 2021.

[19] Windows sysinternals. https://docs.microsoft.com/en-us/sysinternals/, 2021.

[20] S. Ansari, N. Wadhwa, R. Garg, and J. Chen. Wireless software synchronization of multiple distributed cameras. In *2019 IEEE International Conference on Computational Photography (ICCP)*, pages 1–9, 2019.

[21] J. Coleman, S. Almalih, A. Slota, and Y.-H. Lee. Emerging cots architecture support for real-time tsn ethernet. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, pages 258–265. ACM, 2019.

[22] T. Cooklev, J. C. Eidson, and A. Pakdaman. An implementation of ieee 1588 over ieee 802.11 b for synchronization of wireless local area network nodes. *IEEE Transactions on Instrumentation and Measurement*, 56(5):1632–1639, 2007.

[23] S. D'souza, H. Koehler, A. Joshi, S. Vaghani, and R. R Rajkumar. Quartz: <i>time-as-a-service</i> for coordination in geo-distributed systems. In *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*, SEC '19, page 264–279, New York, NY, USA, 2019. Association for Computing Machinery.

[24] R. Exel. Clock synchronization in ieee 802.11 wireless lans using physical layer timestamps. In *2012 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication Proceedings*, pages 1–6, 2012.

[25] B. Ferencz and T. Kovácsházy. Hardware assisted cots ieee 1588 solution for x86 linux and its performance evaluation. In *2013 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication (ISPCS) Proceedings*, pages 47–52, 2013.

[26] P. Ferrari, A. Flammini, S. Rinaldi, A. Bondavalli, and F. Brancati. Experimental characterization of uncertainty sources in a software-only synchronization system. *IEEE Transactions on Instrumentation and Measurement*, 61(5):1512–1521, 2012.

[27] M. Ibrahim, H. Liu, M. Jawahar, V. Nguyen, M. Gruteser, R. Howard, B. Yu, and F. Bai. Verification: Accuracy evaluation of wifi fine time measurements on an open platform. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, pages 417–427. ACM, 2018.

[28] J. Kannisto, T. Vanhatupa, M. Hannikainen, and T. Hamalainen. Software and hardware prototypes of the ieee 1588 precision time protocol on wireless lan. In *2005 14th IEEE Workshop on Local and Metropolitan Area Networks*, pages 1–6, 2005.

[29] J. Kannisto, T. Vanhatupa, M. Hännikäinen, and T. D. Hämäläinen. Precision time protocol prototype on wireless lan. In *International Conference on Telecommunications*, pages 1236–1245. Springer, 2004.

[30] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978.

[31] Z. Li, W. Chen, C. Li, M. Li, X.-Y. Li, and Y. Liu. Flight: Clock calibration using fluorescent lighting. In *Proceedings of the 18th annual international conference on Mobile computing and networking*, pages 329–340. ACM, 2012.

[32] R. Lisovỳ, M. Sojka, and Z. Hanzálek. Pci express as a killer of software-based real-time ethernet. In *The 12th International Workshop on Real-Time Networks*, 2013.

[33] A. Mahmood, R. Exel, and T. Sauter. Delay and jitter characterization for software-based clock synchronization over wlan using ptp. *IEEE Transactions on Industrial Informatics*, 10(2):1198–1206, 2014.

[34] A. Mahmood, R. Exel, H. Trsek, and T. Sauter. Clock synchronization over ieee 802.11—a survey of methodologies and protocols. *IEEE Transactions on Industrial Informatics*, 13(2):907–922, 2017.

[35] A. Mahmood, G. Gaderer, H. Trsek, S. Schwalowsky, and N. Kero. Towards high accuracy in ieee 802.11 based clock synchronization using ptp. In *2011 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*, pages 13–18, 2011.

[36] A. Mahmood and T. Sauter. Limitations in implementing wireless clock synchronization using the timing advertisement approach from ieee 802.11. In *2016 IEEE International Instrumentation and Measurement Technology Conference Proceedings*, pages 1–6. IEEE, 2016.

[37] S. K. Mani, R. Durairajan, P. Barford, and J. Sommers. Mntp: enhancing time synchronization for mobile devices. In *Proceedings of the 2016 Internet Measurement Conference*, pages 335–348. ACM, 2016.

[38] P. Membrey, D. Veitch, and R. K. Chang. Time to measure the pi. In *Proceedings of the 2016 Internet Measurement Conference*, pages 327–334. ACM, 2016.

[39] D. L. Mills. Improved algorithms for synchronizing computer network clocks. *IEEE/ACM transactions on Networking*, 3(3):245–254, 1995.

[40] R. Neugebauer, G. Antichi, J. F. Zazo, Y. Audzevich, S. López-Buedo, and A. W. Moore. Understanding pcie performance for end host networking. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 327–341, 2018.

[41] H. Rahul, H. Hassanieh, and D. Katabi. Sourcesync: a distributed wireless architecture for exploiting sender diversity. *ACM SIGCOMM Computer Communication Review*, 41(4):171–182, 2011.

[42] J. Ridoux, D. Veitch, and T. Broomhead. The case for feed-forward clock synchronization. *IEEE/ACM Transactions on Networking (TON)*, 20(1):231–242, 2012.

[43] R. Schöne, D. Molka, and M. Werner. Wake-up latencies for processor idle states on current x86 processors. *Computer Science-Research and Development*, 30(2):219–227, 2015.

[44] K. B. Stanton. Distributing deterministic, accurate time for tightly coordinated network and software applications: Ieee 802.1as, the tsn profile of ptp. *IEEE Communications Standards Magazine*, 2(2):34–40, 2018.

[45] Y.-H. Wei, Q. Leng, S. Han, A. K. Mok, W. Zhang, and M. Tomizuka. Rt-wifi: Real-time high-speed communication protocol for wireless cyber-physical control applications. In *2013 IEEE 34th Real-Time Systems Symposium*, pages 140–149. IEEE, 2013.

[46] M. Weiss. Telecom requirements for time and frequency synchronization. In *National Institute of Standards and Technology (NIST), USA,[Online]: www. gps. gov-/cgsic/meetings/2012/weiss1. pdf*, 2012.

[47] Z. Yang, J. Zhang, K. Tan, Q. Zhang, and Y. Zhang. Enabling tdma for today's wireless lans. In *2015 IEEE Conference on Computer Communications (INFOCOM)*, pages 1436–1444. IEEE, 2015.