Delay and Jitter Characterization for Software-Based Clock Synchronization Over WLAN Using PTP

Aneeq Mahmood, Reinhard Exel, *Member*, *IEEE*, and Thilo Sauter, *Fellow*, *IEEE*

***Abstract*—In distributed systems, clock synchronization performance is hampered by delays and jitter accumulated not only in the network, but also in the timestamping procedures of the devices being synchronized. This is particularly critical in software timestamp-based synchronization where both software- and hardware-related sources contribute to this behavior. Usually, these synchronization impairments are collapsed into a black-box performance figure without quantifying the impact of each individual source, which obscures the picture and reduces the possibility to find optimized remedies. In this study, for the first time, the individual sources of delay and jitter are investigated for an IEEE 802.11 wireless local area network (WLAN) synchronization system using the IEEE 1588 protocol and software timestamps. Novel measurement techniques are proposed to quantify the hardware- and software-related delay and jitter mechanisms. It is shown that the delays and their associated jitter originate from both the WLAN chipset and the host computer. Moreover, the delay from the chipset cannot be considered symmetric and any such assumption inevitably leads to a residual offset, and thus to synchronization inaccuracy. Therefore, a calibration-based approach is proposed to compensate for these delays and to improve the performance of WLAN synchronization. Experimental results show that with optimal error compensation, a similar synchronization performance as software-based synchronization in Ethernet networks can be achieved.**

***Index Terms*—Accuracy, clock synchronization, IEEE 802.11, IEEE 1588, precision time protocol (PTP), precision, software timestamping, wireless local area network (WLAN).**

I. Introduction

THE INCREASED flexibility and low deployment costs offered by IEEE 802.11 wireless local area network (WLAN) [1] have extended its usage from normal offices to distributed networks for various applications such as instrumentation and measurement, industrial automation [2], or even smart grids [3]. Like all networks that originated from the IT world, WLAN *per se* has only modest real-time capabilities, which makes its applicability in automation scenarios questionable at first sight [4]. A way to overcome this deficiency and to coordinate distributed activities in the network is to use clock synchronization. For automation applications, the master/slave-based IEEE 1588 Precision Time Protocol (PTP) [5] is the favored protocol nowadays. The reason for using PTP is mostly its relatively simple structure and its capability of achieving higher synchronization performance than the Network Time Protocol (NTP) [6]. This has led to a broadening acceptance of PTP in industry [7] as evident by various PTP profiles, including the usage over WLAN [8].

The high synchronization performance, achieved by PTP, is through packet timestamping inside the hardware of the network interface card (NIC) after packet transmission and reception. Such timestamps reflect the exact time of arrival or departure of synchronization packets and can help in achieving synchronization precision in the range of a few nanoseconds [9] regardless of the PTP profile being used. This reliance on timestamps can further be explained with the help of Fig. 1, which shows the basic PTP synchronization principle. The master sends a PTP Sync packet at time t_1, which reaches the slave at t_2. Using the timestamp difference $t_2 - t_1$, the slave can synchronize to the master. However, this difference also includes the propagation delay d_{ms} from the master clock to the slave clock. To solve this issue, the slave sends a `Delay_Req` frame at t_3, which reaches the master at t_4 after a propagation delay d_{sm}. The master sends the timestamp t_4 back to the slave using a `Delay_Resp` message. Using the timestamps, the offset between the master and the slave clocks is given as

$$\text{offset} = \frac{(t_2 - t_1) - (t_4 - t_3)}{2} - \frac{d_{\mathrm{ms}} - d_{\mathrm{sm}}}{2}. \qquad (1)$$

If the propagation delay in both directions is symmetric ($d_{\mathrm{ms}} = d_{\mathrm{sm}}$), then (1) shows that the offset can be calculated precisely, given the timestamp are noiseless, which can be ensured by hardware timestamping. However, commercial-off-the-shelf (COTS) wireless devices cannot do hardware timestamp for PTP packets. To overcome this limitation, version 2012 of the IEEE 802.11 [1] has introduced timing measurement action frames, which provide hardware timestamps for application synchronization. However, as of now, such WLAN NICs are not available in the market. The proprietary WLAN devices of [9] can perform hardware timestamping but result in nonconformity with the standard and higher costs than COTS devices. Hence, to maintain standard conformity and avoid large deployment costs, synchronization over COTS WLAN devices is to be materialized using software timestamps, which are drawn by the operating system (OS).

Deploying software-based WLAN synchronization is cost-effective, but suffers from the jitter of software timestamps. The jitter can be reduced if timestamps are drawn as close to the physical layer (PHY), which is in the interrupt service routine (ISR) inside the device driver. This will avoid the jitter coming

Manuscript received September 04, 2013; revised December 03, 2013; accepted January 15, 2014. Date of publication February 03, 2014; date of current version May 02, 2014. Paper no. TII-13-0604.

A. Mahmood is with the Institute of Computer Technology, Vienna University of Technology, 1040 Wien, Austria (e-mail: aneeq.mahmood@tuwien.ac.at).

R. Exel and T. Sauter are with the Centre for Integrated Sensor Systems, Danube University Krems, 3500 Krems an der Donau, Austria.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TII.2014.2304413

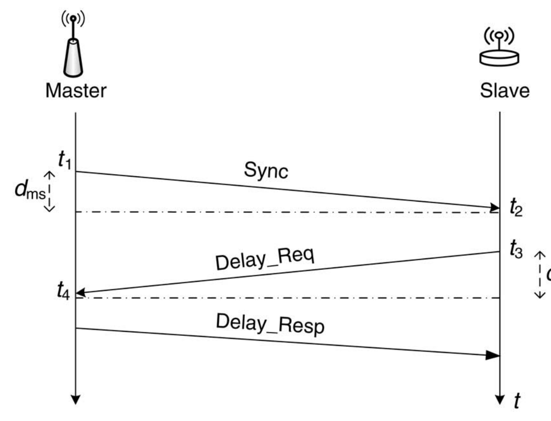


Fig. 1. PTP-based message exchange for synchronization.

from higher layers of the protocol stack and will also circumvent the uncertainty introduced by the channel access scheme of IEEE 802.11. However, the processing done inside the WLAN chipset after transmission and reception, interrupt handling (IH) by the CPU, and drawing of the timestamp by the OS still imparts uncertainty to the actual time of packet reception or transmission. Moreover, the assumption for symmetric path delays cannot always hold because of potentially different propagation delays during transmission, reception, and IH. Consequently, any path-delay asymmetry during propagation will result in a bias, which will be reflected in the offset calculation as indicated by (1).

The goal in this work is to characterize these different individual delays and their associated jitter for software timestamp-based synchronization. This is achieved by proposing, for the first time, the measurement techniques for analyzing delays coming from both the WLAN chipsets and IH and timestamping. For achieving this goal, the open source device *ath5k* drivers [10] for Linux can be used, as they provide direct access to hardware and do not contain any other abstraction layer between the hardware and the OS. The knowledge obtained from this work can then not only be used for synchronizing newly deployed WLANs but can also be used to improve the performance of existing wireless and wired networks, which rely on software timestamping.

II. State of the Art

Owing to the lack of knowledge about the underlying hardware in software timestamp-based synchronization, it is common to treat the system as a black-box, draw timestamps inside the driver or at a higher layer, and focus only on the final synchronization performance as shown in [11] and [12]. Studies such as [13] and [14] analyze either the software timestamping delay or the IH delay in Linux and do not cover other error sources of delay and jitter. A study that diverges from the typical black-box approach is carried out by [15], which analyzes the propagation delay inside the chipset and IH delay for wired synchronization. This is performed by drawing a hardware timestamp from the clock inside the wired NIC upon packet transmission and reception, and then comparing it with the subsequent software timestamp drawn from the same clock inside the device driver. However, this scheme is not practical for COTS WLAN devices, as they cannot perform hardware timestamping for data frames inside the NIC. Based on [15], a similar setup has been proposed in [16] for NICs without hardware timestamping capabilities. The equivalent of timestamping inside the chipset is achieved by using a network analyzer between the master and the slave clock. The performance metric in [16] is an uncertainty index, which essentially provides the combined delay and jitter for the whole proverbial black-box. However, as the role of individual delay and jitter sources is not known, the extent of their contribution toward the final synchronization performance cannot be fully examined in [16]. This study overcomes such limitations by investigating each delay and jitter source involved in software-based synchronization, and also determines the factors on which these error sources depend upon. Moreover, how to quantify the impact of individual delay and jitter sources on the final synchronization accuracy and precision will also be addressed in this work.

The identification of the major delay and jitter sources is done in Section III, Section IV describes the delay contributed by the chipsets, Section V describes the contribution from software components in the system, Section VI analyzes the WLAN synchronization performance, and Section VII provides concluding remarks.

III. Identification of Delay Sources

According to IEEE 1588, the timestamping instant is defined as the moment when the start of frame delimiter (SOF) within the frame (packet) header during transmission and reception passes the “reference plane” of the clock. However, if the timestamp is drawn at an instant other than this designated instant, then the timestamp needs to be corrected, so that it reflects the time when the SOF within the frame would have passed the reference plane of the clock. For interrupt-driven timestamping, the timestamps are drawn inside the ISR after packet transmission or reception. Mapping the software timestamps to the timestamping instant mentioned by IEEE 1588 requires determining all the delay sources that affect packet handling inside the chipset and inside the OS. This inevitably leads to identifying all the delay sources that affect software-based synchronization.

For a simpler analysis in this work, the reference point is shifted from SOF to the instant when the frame check sequence (FCS) at the end of the WLAN frame is sent during transmission and is received during reception. Later on, the WLAN chipset can be calibrated, so that the timestamps reflect the instant when the SOF passes the reference plane of the clock to stay compatible with PTP.

Fig. 2 shows the propagation of a Sync packet from the transmitter (Tx) to the receiver (Rx) and the drawing of software timestamps. The packet, after traversing through the protocol layers, reaches the WLAN chipset and is sent out via the Tx antenna. After a certain delay d_{hw}^{t}, the WLAN chipset raises the hardware interrupt to indicate successful transmission. This delay is the Tx interrupt notification time and depends upon the implementation inside the chipset. Upon noticing the interrupt, the CPU performs a context switch and determines which device has raised the interrupt. Eventually, the OS executes the ISR of the WLAN device, which is located inside the device driver; the

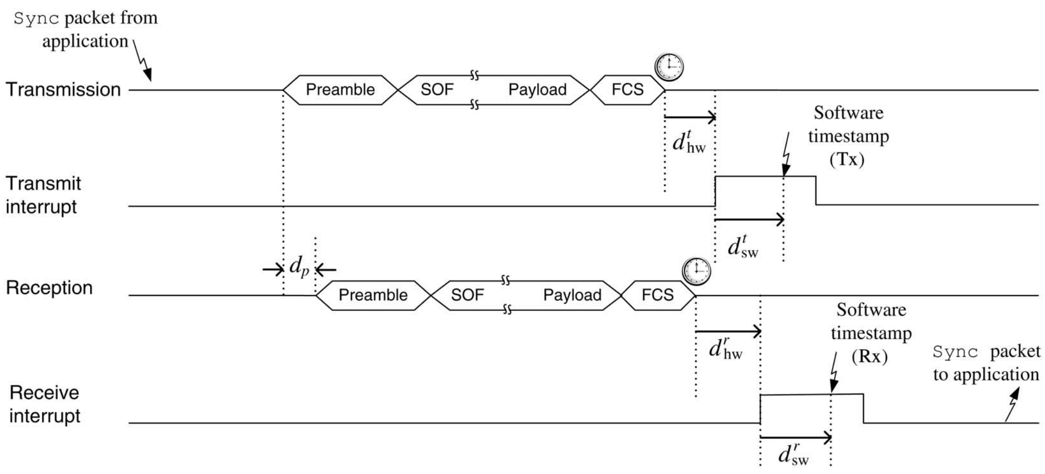


Fig. 2. Delays involved in WLAN synchronization from the beginning of transmission till the software timestamps are drawn at the receiver.

software timestamp is then drawn inside the ISR. The delay at the Tx side for handling the interrupt and drawing the timestamp depends upon the CPU and the type of the OS and is denoted by d^t_{sw}.

At the receiver side, the packet arrives after a propagation delay d_p. The signal is converted to the baseband (BB) and processed by the BB processor, which checks the FCS. The hardware interrupt is asserted after a delay d^r_{hw}. This delay comprises packet processing time inside the PHY and the Medium Access Control (MAC) layers, and packet transfer time from the chipset to the host via the direct memory access (DMA) transfer [17]. Afterward, the interrupt is handled and the software timestamp is drawn after a delay d^r_{sw}. Once the interrupt has been handled, the received SYNC is sent to the application.

Based on this discussion, the total delay from the end of FCS (the reference point for timestamping), until when the software timestamps are drawn at Tx and Rx, can be represented by Δ^t and Δ^r, respectively, and can be expressed as

$$\Delta^t = d^t_{\text{hw}} + d^t_{\text{sw}} \tag{2}$$

and

$$\Delta^r = d^r_{\text{hw}} + d^r_{\text{sw}}. \tag{3}$$

For synchronization, the delays Δ^t and Δ^r need to be determined, which will map the software timestamp to the end of the FCS. As the delays from the right-hand side of both (2) and (3) are not fixed, they can be treated as random variables. Moreover, as the delays occurring inside the WLAN chipset (d^t_{hw}, d^r_{hw}) are independent of the delays inside the host personal computer (PC) (d^t_{sw}, d^r_{sw}), their first- and second-order statistics can be used to express the total delay and jitter. Hence, if μ represents the sample mean and σ represents the jitter, then

$$\mu_{\Delta^j} = \mu_{d^j_{\text{hw}}} + \mu_{d^j_{\text{sw}}} \tag{4}$$

$$\sigma_{\Delta^j} = \sqrt{\sigma^2_{d^j_{\text{hw}}} + \sigma^2_{d^j_{\text{sw}}}} \tag{5}$$

where $j \in \{t, r\}$. In Sections IV and V, the mechanism to measure the hardware- and software-based delays in transmission and reception are discussed.

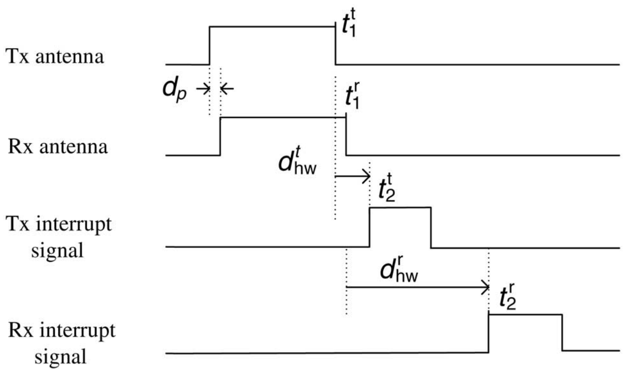


Fig. 3. Sequence of events during packet transmission and reception.

IV. Measurement of Hardware Delays

Fig. 3 highlights the hardware delays during packet transmission and reception chronologically. The transmission finishes at time t^t_1 and the hardware interrupt is raised at t^t_2. Similarly, reception finishes at time t^r_1 and the interrupt is raised at t^r_2. From Fig. 3, the interrupt notification delay at the receiver can be obtained from

$$d^r_{\text{hw}} = t^r_2 - t^t_2 + d^t_{\text{hw}} - d_p. \tag{6}$$

The difference $t^r_2 - t^t_2$ can be measured by feeding the hardware interrupt signal at the transmitter and the receiver into an oscilloscope. The propagation delay d_p can be considered constant for static receivers or for receivers with limited mobility and in line of sight (LOS) with the transmitter. The delay d^r_{hw} can then be calculated using (6) if d^t_{hw} is known. The method to measure d^t_{hw} (and the resulting d^r_{hw}) is discussed below.

A. Delay at the Tx Side

At the transmitter side, after the last bits of the frame are successfully sent on the channel, the PHY informs the MAC layer about the end of transmission. Afterward, the power amplifiers are ramped down by the PHY, and the WLAN chipset changes from transmit to receive mode. The hardware

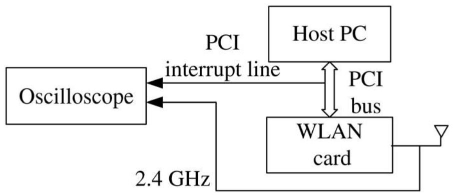


Fig. 4. Setup for measuring transmit interrupt notification delay $d^t_{\rm hw}$.

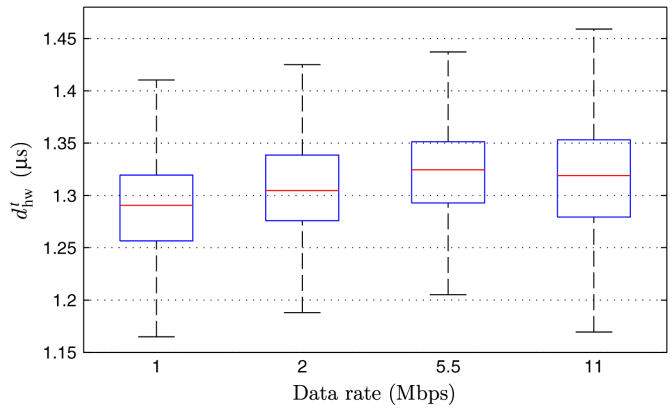


Fig. 5. Value of $d^t_{\rm hw}$ for IEEE 802.11b data rates.

interrupt is raised at some point between the end of transmission and switching the chipset into receiver mode. The exact instant of interrupt assertion depends upon the hardware implementation and may change from vendor to vendor. However, as the interrupt notification delay at the transmitter does not involve handling of the frame, it should not vary regardless of the modulation scheme or the data rate the packet has been sent at.

To verify this statement and to measure $d^t_{\rm hw}$, a simple measurement setup is proposed, which is shown in Fig. 4. An Atheros WLM54AGP23 WLAN chipset, housed inside an Axiomtek industrial personal computer (IPC), is used as the transmitter. The IPC uses an Intel Core2 Processor T7400 (2.16 GHz) as its CPU. The hardware interrupt signal from the peripheral component interconnect (PCI) bus of the computer is fed into a digital oscilloscope. The antenna of the WLAN chipset is replaced by a cable, which is then fed into the oscilloscope that samples the incoming signal at 10 GSamples/s. The delay $d^t_{\rm hw}$ is calculated inside the oscilloscope by monitoring the signal coming from the WLAN antenna and the hardware interrupt from the PCI bus. To determine the impact of various modulation schemes and data rates, $d^t_{\rm hw}$ is measured for data rates supported by IEEE 802.11b.

Fig. 5 displays the result of $d^t_{\rm hw}$ for various data rates as boxplots. The mean value for $d^t_{\rm hw}$ is 1.66 µs with a standard deviation of 46 ns for all data rates with the Atheros chipset used in this study. Other chipsets may yield a different mean delay and standard deviation. This result supports the assertion that time to raise the interrupt at Tx side is independent of the data rate and only depends on the hardware implementation inside the chipset.

B. Delay at the Rx Side

Once $d^t_{\rm hw}$ has been determined using the setup from Fig. 4, $d^r_{\rm hw}$ can be determined by finding out the time difference between the hardware interrupt lines at Tx and Rx and using (6). However, unlike $d^t_{\rm hw}$, $d^r_{\rm hw}$ is not constant and its value depends upon several factors that are described below.

1) *Data rate (R):* The value of $d^r_{\rm hw}$ is dependent upon the modulation scheme (data rate) used during transmission. Different modulation schemes will result in different demodulation and processing delays inside the chipset, thus affecting $d^r_{\rm hw}$ accordingly.
2) *Capacity of the interface between COTS WLAN card and the host (B):* In modern computers, the transfer of the received packet from the chipset to the host is carried out via standardized interfaces which can perform either serial or parallel data transfer. Commonly used interfaces for communication with networking peripherals are PCI, PCIExpress (PCIe), and Universal Serial Bus (USB). Each interface has its own capacity and the faster the interface, the quicker the exchange of data from the WLAN hardware to the host and the subsequent assertion of the receive interrupt.
3) *Packet size:* For a fixed bus interface between the WLAN card and the host, the larger the packet size, the longer it will take to transfer it to the host. Consequently, the receive interrupt will be generated later for larger packets than for smaller packets, and it will also be reflected in the value of $d^r_{\rm hw}$.
4) *DMA burst size and buffer threshold:* The DMA controller, residing on the WLAN card, is responsible for transferring the received packet from the chipset to the appropriate location in the host memory. During the course of the packet reception, the data are demodulated and decoded and saved to an on-chip DMA first in-first out (FIFO) buffer which has a storage capacity of n bytes. However, to avoid buffer overflow and loss of incoming data, the buffer has a threshold of α bytes as shown in Fig. 6. When the incoming frame size is smaller than α, the threshold is not exceeded, and the buffer is flushed when the whole packet comes into it followed by the hardware interrupt. However, if the incoming packet is bigger than α, a DMA burst with burst size β will occur that will send β bytes from the buffer to the host to make more room for the incoming bits. Later, when the rest of the packet arrives into the buffer, the buffer will be flushed. Because of the earlier DMA burst, the data to be flushed inside the buffer are smaller than α. Consequently, the delay for packet sizes greater than α will be smaller than for sizes approaching to α. Therefore, the size (threshold) of the FIFO buffer and the burst size will also influence $d^r_{\rm hw}$.

From this discussion, it can be concluded that the value of $d^r_{\rm hw}$ remains constant for a fixed data rate. Also, if the interface between the card and the host is fast, $d^r_{\rm hw}$ will be smaller. The increasing packet size results in increasing $d^r_{\rm hw}$ until α is reached,

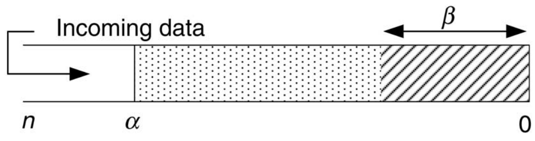


Fig. 6. Layout of the DMA buffer.

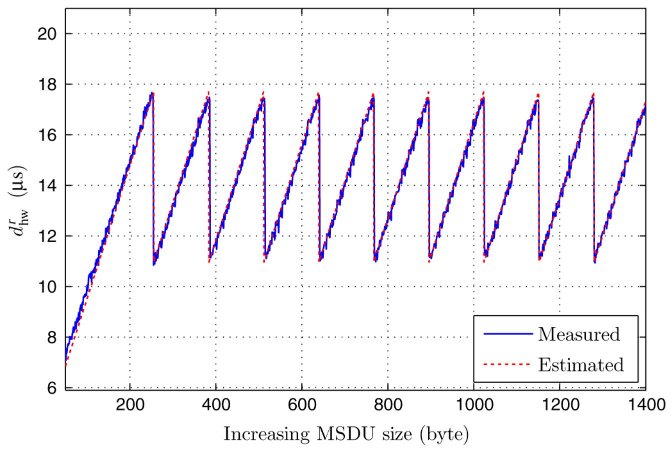


Fig. 7. Measured and estimated d^r_{hw} at 1 Mbps for increasing packet sizes.

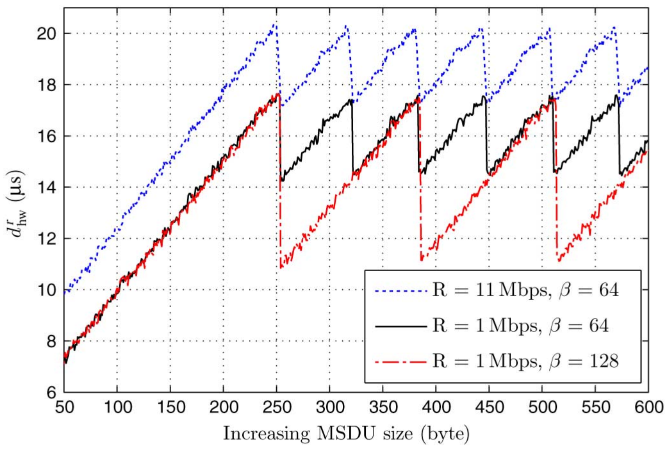


Fig. 8. d^r_{hw} for different data rates and DMA burst sizes.

which will result in a DMA burst, and d^r_{hw} will decrease. Now if the packet size is again increased, d^r_{hw} will increase again until α is about to be usurped again. For such packets, there will be a second DMA burst and the resulting value of d^r_{hw} will drop again. As a result, if the packet size keeps increasing, d^r_{hw} will follow a sawtooth pattern shown in Fig. 7. Mathematically, d^r_{hw} can be estimated by the relation

$$d^r_{\mathrm{hw}} = \begin{cases} L + \frac{m}{B}, & s \le m \le \alpha \\ L + \frac{1}{B}\left(\alpha + m - \beta\left(1 + \left\lfloor \frac{m}{\beta} \right\rfloor\right)\right), & m > \alpha \end{cases} \quad (7)$$

where m is the size of the packet in bytes. Its minimum value is the size of a standard Sync packet s, and L is the PHY and the MAC processing delay in the WLAN hardware; its value changes with the modulation schemes and the resulting data rates but is assumed constant for one specific data rate R.

While some of the parameters in (7) can be obtained from the device driver, L and α are not available through the driver or the datasheet of the chipset which makes the estimation of d^r_{hw} from (7) infeasible. Thus, a measurement-based approach using (6) can be employed for this purpose. To measure d^r_{hw}, a simple wireless setup is established using two static computers 1 m apart; one computer acts as an access point (AP) and the other as a slave wireless node. Each computer uses the same hardware as used for the setup in Fig. 4. The delay d^t_{hw} is already obtained for different IEEE 802.11b data rates and d_p is also fixed to 3.3 ns. The value $t^r_2 - t^t_2$ is obtained by sending linearly increasing MAC service data units (MSDUs) from the AP to the node and by connecting the interrupt lines to the oscilloscope and calculating their difference.

Fig. 7 shows d^r_{hw} at 1 Mbps with DMA burst size of 128 bytes from the measurement (solid line) and also compares it against calculated values (dashed line) obtained from (7). The values of L and $1/B$ for (7) are not known beforehand are obtained from the measured d^r_{hw}. Fig. 7 indicates that d^r_{hw} follows the pattern predicted by (7).

Fig. 8 shows the impact of varying data rates and DMA burst sizes on d^r_{hw}. When the data rate changes from 1 Mbps to higher rates, more complex modulation and coding schemes are used, hence increasing the processing time inside the chipset and resulting in a higher value of L. This is exhibited in Fig. 8 where d^r_{hw} for 11 Mbps (blue, solid line) is approximately 2 µs greater than at 1 Mbps (black dashed-dotted line). Moreover, using a larger β value leads to more space inside the DMA buffer after a DMA burst for packet sizes greater than α. This results in smaller values of d^r_{hw} for larger burst sizes than for small bursts. The red dotted line in Fig. 8 exhibits this behavior for $\beta = 128$ when compared with the black dash dotted line which displays d^r_{hw} for $\beta = 64$.

Based on this discussion, it can be seen that if d^r_{hw} in master–slave direction is different than in slave–master direction, a path-delay asymmetry will exist which will result in a synchronization bias. For example, if the data rate in either direction is not the same because of varying radio conditions or rate adaptation mechanisms [18], L will be different in both directions resulting in a bias. Similarly, the introduction of a type, length, value (TLV) field in either `Sync` or `Delay_Req` will change the packet size in one direction. Thus, d^r_{hw} in that direction will be larger causing path-delay asymmetry. Finally, different values of α and β at the master and slave will also result in the same behavior.

Hence, to remove any synchronization bias because of delays from the WLAN hardware, it is proposed to calibrate the hardware delays for all data rates, plausible packet sizes, and DMA buffer properties, and store this information in a lookup table (LUT) inside the device driver. The ingress and the egress timestamps can then be compensated for hardware delays using appropriate d^t_{hw} and d^r_{hw} values from the LUT. This will lead to the removal of any potential sources of synchronization bias from the WLAN hardware.

Although the measurement setup proposed in this section uses only Atheros chipsets over the PCI bus, the setup is generic in nature and can be easily extended to COTS chipsets from any other vendor. As the interrupt signal from the hardware is an integral part of the measurement setup and calibration, the only prerequisite is that other COTS WLAN cards should be able to perform interrupt-based communication. The subsequent interrupt signal can be extracted directly from the data bus or from the interrupt controller on the motherboard of the host PC. Hence, the measurement setup is not bound to be used only with the PCI bus.

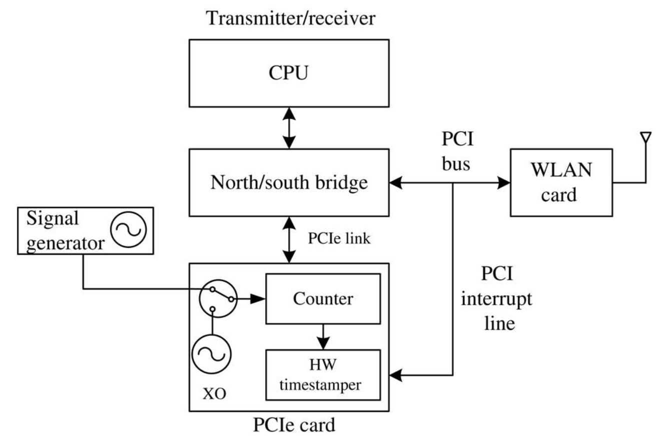


Fig. 9. Setup for measuring interrupt handling and timestamping delay at the transmitter/receiver.

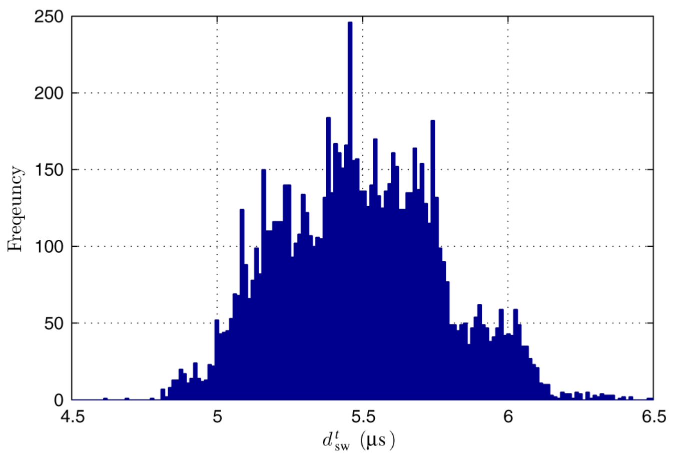


Fig. 10. Interrupt handling and timestamping delay at the transmitter.

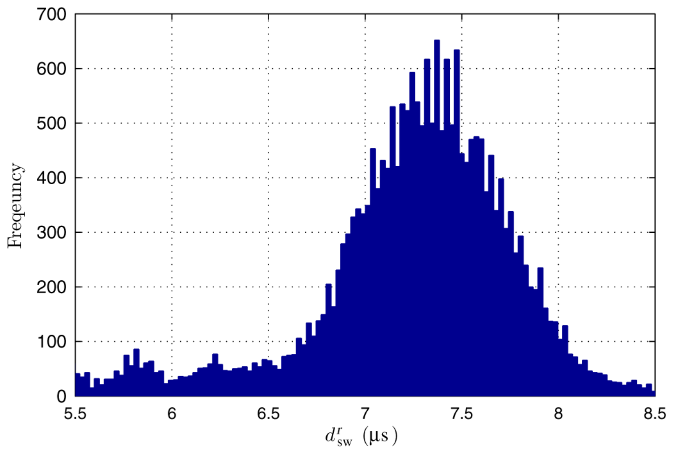


Fig. 11. Interrupt handling and timestamping delay at the receiver.

V. IH AND TIMESTAMPING DELAY

The delays associated with IH and timestamping (d^t_{sw}, d^r_{sw}) have been previously modeled using commonly used probability distribution functions (PDFs) such as Gaussian-Bernoulli [19] and Exponential [20] distributions. However, these delays depend upon several factors such as the interrupt controller, the OS, capabilities of the CPU, load on the CPU, etc., and therefore the resulting composite PDFs diverge significantly from these generic PDFs. Therefore, a measurement setup is proposed to evaluate the delay distribution functions at both the Tx and the Rx side.

The proposed measurement setup is similar to the one used in [15] in the sense that a single clock source residing on the NIC is used for both hardware and software timestamping. However, as wireless NICs cannot perform hardware timestamping for PTP packets, an extra peripheral device is used which can generate a hardware timestamp upon detecting an external event. The external event for hardware timestamping is created by feeding the hardware interrupt line from the PCI bus to the peripheral device. Hence, whenever a hardware interrupt is raised, the peripheral device draws a hardware timestamp from its on-board clock. Later on, when the subsequent ISR is executed, another timestamp is drawn from the same clock—albeit by software means. The difference between the hardware and software timestamps at the Tx side and the Rx side gives d^t_{sw} and d^r_{sw}, respectively.

Fig. 9 shows the measurement setup being used to measure d^t_{sw} or d^r_{sw}. The peripheral device at the Tx and the Rx side for hardware timestamping is Syn1588 PCIe card [21] from Oregano Systems. This PCIe card has a physical port, which can detect external signals, and a timestamping unit, which can draw a hardware timestamp from a free-running clock tick counter with a timestamping resolution of 10 ns, upon occurrence of the external signal on its port. The interrupt line from the PCI bus is connected to this port. The OS draws the software timestamps by reading the free running counter using the *cyclecounter* data structure provided by Linux. The frequency drift and phase noise, coming from the crystal oscillators (XOs) on the PCIe cards, can influence the values of d^t_{sw} and d^r_{sw}. To avoid this, the counter on PCIe card at both the Tx and the Rx side is driven by a Tektronix AFG 3252 function generator with minimal frequency drift and phase noise. The CPU and the WLAN cards for the measurement setup in Fig. 9 are the same as used to measure hardware delays in Section IV. A similar setup is proposed by the authors in [22], but the current setup avoids the errors from the XOs placed on the PCIe card.

Figs. 10 and 11 provide the histograms for the d^t_{sw} and d^r_{sw}, respectively. These delays are affected by the arrival of interrupts of higher priority than for WLAN communication and, hence, can spread over several microseconds. The mean delay at the Rx side is 7.23 μs with a standard deviation of 0.58 μs, which is higher than the delay of 5.4 μs at the Tx side with a standard deviation of 0.31 μs. The relatively higher delay at the receiver can be attributed to the operation of the memory controller such as north/south bridge between the CPU and other peripheral devices. The memory controller overlooks the transfer of the packet from the wireless NIC to the host. Any delay occurring in the packet transfer over the north/south bridge after reception hinders the drawing of the software timestamp, which results in a higher d^r_{sw} as compared to d^t_{sw}. Hence, the CPU architecture and processing capabilities, along with the OS, contribute toward the IH delay after packet transmission and reception.

TABLE I
DELAY AND JITTER IN TRANSMISSION AND RECEPTION

	d^t_{hw}	d^t_{sw}	d^r_{hw}	d^r_{sw}
Mean, μ	1.31 µs	5.4 µs	8.9 µs	7.23 µs
Jitter, σ	46 ns	310 ns	110 ns	580 ns

VI. PERFORMANCE ANALYSIS

Table I summarizes the different Tx and Rx delays that have been measured during various measurement setups using Axiomtek IPCs, which employ Intel Core2 Processor T7400 (2.16 GHz) as CPUs, Atheros WLM54AGP23 WLAN chipsets, and Linux OS. The value of d^r_{hw} has been obtained using (6) for a 94-byte PTP `Sync` packet at 1 Mbps with fixed propagation delay. The value of d^r_{hw} is fixed for all IEEE 802.11b data rates as shown in Fig. 5.

Table I also shows that the jitter coming from the software components is larger than that coming from WLAN chipset. However, depending upon the packet size and data rate, the delay occurring inside the WLAN chipset can be larger than the actual propagation delay on the wireless channel. Thus, delay asymmetry inside the chipsets is more critical than any asymmetry originating from variations in the channel. Consequently, for software timestamping-based synchronization, the jitter can be attributed primarily to the CPU and the OS while the synchronization bias comes from the WLAN chipsets.

Its should be noted that Table I encompasses all the delays in (2) and (3), which are required to map the software timestamps to the reference timestamping instant, which is the end of FCS. Hence, by calibrating the WLAN chipset and measuring the IH and timestamping delays at the Tx and the Rx, the bias can be totally removed without using any `Delay_Req` messages, provided the propagation delay d_p does not change. If the WLAN devices are moving and d_p is not constant, then `Delay_Req` messages should be used. However, even in this case, the propagation delay for a `Sync` cannot always be presumed equal to a `Delay_Req` message because of potentially different packet sizes and/or data rates. Thus, delays inside the chipset still need to be compensated accordingly to make sure that propagation delays in either direction are symmetric. Hence, the compensated delays from the LUT mentioned in Section IV are required regardless of whether `Delay_Req` messages are being used or not.

Finally, Table I has been created only for IEEE 802.11b data rates. In case IEEE 802.11a/g/n rates are being used, delays attributed to hardware will change while delays related to IH and timestamping will remain unchanged. In particular, d^r_{hw} will increase because of more complex receive operations and demodulation schemes being employed in the receiver [1]. Additionally, depending upon the frame size and data rate in IEEE 802.11a/g/n mode, the frame is extended by adding *pad bits* so that the total size of the frame without the preamble bits and PHY header is a multiple of number of data bits per orthogonal frequency-division multiplexing (OFDM) symbol as described in [1]. This leads to a larger DMA transfer time and larger value of d^r_{hw} as compared to frames using IEEE 802.11b data rates. Hence, if IEEE 802.11a/g/n data rates are being used without proper calibration, path-delay asymmetry can lead to larger delays than for IEEE 802.11b data rates.

A. Testbed Setup

To highlight the significance of calibrating hardware- and software-based delays for software-based WLAN clock synchronization, a testbed is established using two WLAN devices acting as the master (AP) and the slave (node). The CPUs and WLAN chipsets, used for the master and the slave, are the same as used for the values for Table I. The master and the slave devices are stationary and are in LOS with each other; the synchronization interval is set to 1 s. As no wireless specific profile is prescribed by PTP, a generic multicast-based Layer 3 PTP profile is used for synchronization. This will help exploit the broadcast nature of the wireless channel as the multicast `Sync` packets can be sent simultaneously to all the nodes in the network; these multicast packets do not request the node to send IEEE 802.11 acknowledgement (`ACK`) frames, which avoids the problem of missed `ACK` frames and retransmissions. The control loop settings for the slave device are obtained from [22], which ensures jitter minimization by trusting more on the node's oscillator than on the timestamps. This also provides an intrinsic fault tolerance in the system against `Sync` frames being lost on the channel as the nodes are relying relatively less on the software timestamps for staying synchronized to the AP as highlighted in [22].

For scenarios where the efficient bandwidth usage is sought, beacons can be used to act as both `Sync` and `Follow_up` frames as suggested in [11]. However, a typical Layer 3 PTP implementation is used in this work, where `Follow_up` packets are used to transmit the accurate transmit timestamps to the node. The synchronization accuracy is verified by comparing the 1 pulse per second (PPS) signals from the master and the slave clock. The average difference between the 1 PPS signals provides the mean clock offset while the standard deviation from mean offset presents the jitter. Because the software-based synchronization is affected by the CPU load and the network load, the synchronization performance has been analyzed with and without load on the system in the following discussion.

B. Performance Without Load

For this measurement, the master and the slave devices are under no load and the only activity they perform is to exchange PTP messages. Three separate cases are analyzed with this setup to compare the performance of a standard PTP implementation with the one using calibrated delays. These cases involve sending `Sync` messages from the master to the slave at 1 Mbps and:

1) sending `Delay_Req` messages also at 1 Mbps;
2) sending `Delay_Req` messages at 11 Mbps to highlight propagation delay asymmetry because of data rate;
3) using calibrated delays from the LUT for both Tx and Rx sides, yet with no `Delay_Req` messages.

The steady-state synchronization performance in all three cases is analyzed using histograms which are fitted to Gaussian distributions and are shown in Fig. 12. The mean synchronization offset with the `Sync` and the `Delay_Req` at the same rate is 59 ns with a jitter of 460 ns. However, when the `Delay_Req` messages are sent at 11 Mbps, the master–slave delay and the slave–master

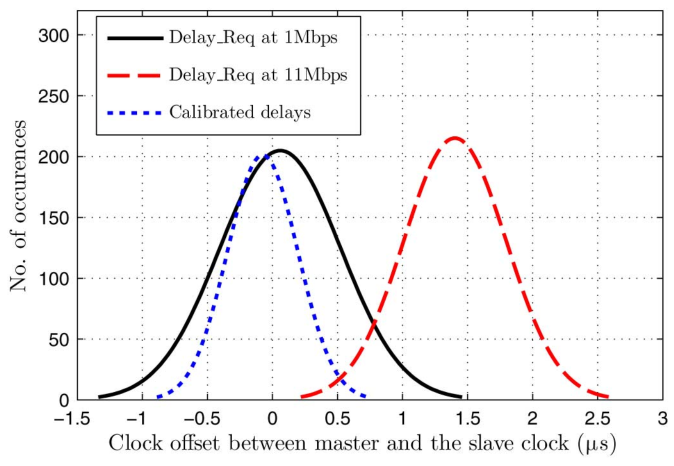


Fig. 12. Synchronization performance under nominal load.

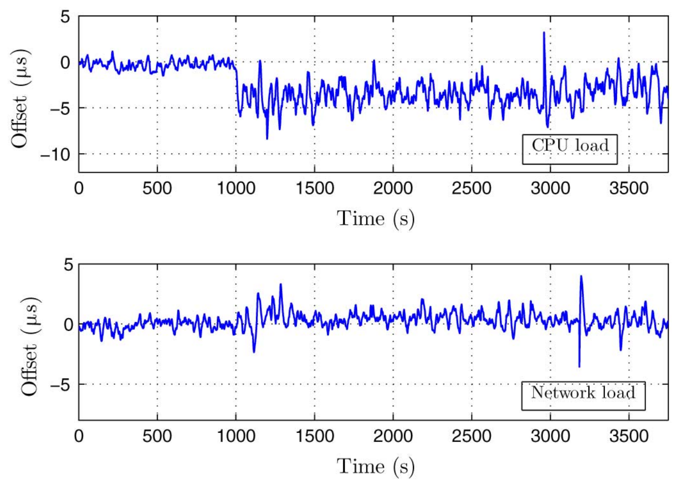


Fig. 13. Synchronization performance under CPU and network load.

delay are no longer symmetric, which results in a synchronization bias of 1.4 μs. Hence, there is a need to ensure that either `Sync` and `Delay_Req` are always sent at the same data rate or the propagation delay inside the chipset is accordingly compensated using LUTs. Similar behaviors can be expected if either `Sync` or `Delay_Req` has additional TLV fields.

The dotted trace "calibrated delays" in Fig. 12 shows that the mean offset, when no `Delay_Req` are used, is comparable to the first case when the `Delay_Reqs` are sent at the same rate. The jitter of the former (320 ns) is smaller than the one of the latter (460 ns). The jitter increases when using `Delay_Reqs` stems from the fact that additional jitter is introduced in the synchronization system by the additional software timestamps required to compensate for the path delay. This is not the case for one-way synchronization using only Sync messages. Consequently, one-way together with the proposed calibration for hardware and software delays offers a better better synchronization precision than two-way synchronization using `Delay_Req` messages.

C. Performance Under Load

The steady-state synchronization performance under load is investigated using only `Sync` messages with calibrated delays. Both master and slave devices are put under CPU load using the *stress* command as also done in [16]. This command generates 100% CPU load by performing mathematical operations, data transfers to and from the hard disk, and I/O operations. The network load is created by directing all the traffic from a loaded 100 Mbps network switch to the wired interface of the master. The wired interface is bridged with the wireless interface of the AP, and hence all the traffic is subsequently directed to the slave. Putting the extra network traffic load on the system results in higher CPU utilization than normal, which is recorded to be 15%.

For both types of load, synchronization takes place for 1000 s without any load and then the load is introduced for 45 min. The upper plot in Fig. 13 shows the difference between the 1 PPS pulses of the master and slave clock and highlights the impact of CPU load on synchronization. Under the impact of load, the synchronization offset moves from 109 ns to −3.25 μs. Similarly, the jitter under CPU load turns out to be 1.6 μs as compared to 360 ns under no load. As shown in [14], the IH delay can jump to really high values because of the non-realtime nature of Linux; this results in large spikes in clock offset. Such behavior does not show up in during this experiment, but may eventually show up in a longer measurement.

The lower plot in Fig. 13 shows the impact of network traffic on the channel for 45 min after 1000 s of only synchronization traffic. Whereas the mean offset is 109 ns before the network load, the offset becomes 316 ns in the presence of network load. Similarly, the jitter increases from 360 ns to 1.26 μs under network load. These measurements reveal that synchronization based on software timestamps suffers from the load on the system. Nevertheless, the steady-state delay and jitter under load remain under 10 μs, which is comparable to PTP over Ethernet with software timestamps [13].

VII. Conclusion

In this paper, an analysis of various delay and jitter sources involved in software timestamping-based clock synchronization over WLAN has been carried out. Instead of treating the underlying system between the egress and the ingress timestamps as a black-box, this work has proposed measurement techniques to quantify the delay and the jitter not only from IH and timestamping but also from the WLAN chipset as well. The measurements have demonstrated that a major source of jitter in synchronization comes from IH, while the delay inside the WLAN chipset during reception plays a significant role for the synchronization bias. Additionally, it has been analyzed and experimentally shown that the path delay cannot be assumed symmetric for varying packet sizes and data rates. To remove such asymmetries, a calibration-based approach is used where both hardware and the software delays for different packet sizes and data rates are measured beforehand and stored in a lookup table for correcting the software timestamps to the timestamp reference plane. The synchronization results have shown that this approach can deal with path delay asymmetries and removes any clock offset bias. The proposed solution provides a better synchronization precision than using plain IEEE 1588 over WLAN and offers a performance comparable to software timestamp-based synchronization over Ethernet.

References

[1] *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 Standard, 2012.

[2] G. Gaderer, T. Sauter, F. Ring, and A. Nagy, "A novel, wireless sensor/actuator network for the factory floor," in *Proc. IEEE Sens. Conf.*, Waikoloa, HI, USA, Nov. 2010, pp. 940–945.

[3] P. Parikh, T. Sidhu, and A. Shami, "A comprehensive investigation of wireless LAN for IEC 61850 based smart distribution substation applications," *IEEE Trans. Ind. Informat.*, vol. 9, no. 3, pp. 1466–1476, Aug. 2013.

[4] G. Cena, L. Seno, A. Valenzano, and C. Zunino, "On the performance of IEEE 802.11e wireless infrastructures for soft-real-time industrial applications," *IEEE Trans. Ind. Informat.*, vol. 6, no. 3, pp. 425–437, Aug. 2010.

[5] *IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*, IEEE Standard 1588, 2008.

[6] G. Cena, I. Bertolotti, S. Scanzio, A. Valenzano, and C. Zunino, "Synchronize your watches: Part I: General-purpose solutions for distributed real-time control," *IEEE Ind. Electron. Mag.*, vol. 7, no. 1, pp. 18–29, Mar. 2013.

[7] A. Moreno-Munoz, V. Pallares-Lopez, J. Gonzalez de la Rosa, R. Real-Calvo, M. Gonzalez-Redondo, and I. Moreno-Garcia, "Embedding synchronized measurement technology for smart grid development," *IEEE Trans. Ind. Informat.*, vol. 9, no. 1, pp. 52–61, Feb. 2013.

[8] Q. Honglei, Q. Changquan, C. Li, and C. Yang, "Wireless LXI bus clock synchronization and triggering design," *IEEE Trans. Instrum. Meas.*, vol. 59, no. 9, pp. 2420–2430, Sep. 2010.

[9] R. Exel, "Clock synchronization in IEEE 802.11 wireless LANs using physical layer timestamps," in *Proc. IEEE Symp. Precis. Clock Synch.* (ISPCS), San Francisco, CA, USA, Sep. 2012, pp. 1–6.

[10] D. Dujovne, T. Turletti, and F. Filali, "A taxonomy of IEEE 802.11 wireless parameters and open source measurement tools," *Commun. Surveys Tuts.*, vol. 12, no. 2, pp. 249–262, Apr. 2010.

[11] A. Mahmood, G. Gaderer, H. Trsek, S. Schwalowsky, and N. Kerö, "Towards high accuracy in IEEE 802.11 based clock synchronization using PTP," in *Proc. IEEE Symp. Precis. Clock Synch.* (ISPCS), Munich, Germany, Sep. 2011, pp. 13–18.

[12] J. Kannisto, J. Kannisto, T. Vanhatupa, T. Vanhatupa, M. Hannikainen, M. Hannikainen *et al.*, "Software and hardware prototypes of the IEEE 1588 precision time protocol on wireless LAN," in *Proc. 14th IEEE Workshop Local Metrop. Area Netw.* (LANMAN). Crete, Greece, 2005, pp. 1–6.

[13] J. Ridoux and D. Veitch, "Ten microseconds over LAN, for free (extended)," *IEEE Trans. Instrum. Meas.*, vol. 58, no. 6, pp. 1841–1848, Jun. 2009.

[14] P. Regnier, G. Lima, and L. Barreto, "Evaluation of interrupt handling timeliness in real-time Linux operating systems," *Oper. Syst. Rev.*, vol. 42, no. 6, pp. 52–63, 2008.

[15] H. Weibel and D. Bchaz, "Implementation and performance of time stamping techniques," in *Proc. Conf. IEEE 1588 Std. Precis. Clock Synch. Protocol Netw. Meas. Control Syst.*, Gaithersburg, MD, USA, 2014, pp. 1–7.

[16] P. Ferrari, A. Flammini, S. Rinaldi, A. Bondavalli, and F. Brancati, "Experimental characterization of uncertainty sources in a software-only synchronization system," *IEEE Trans. Instrum. Meas.*, vol. 61, no. 5, pp. 1512–1521, May 2012.

[17] Z. Dittia, G. Parulkar, and J. R. Cox, "The APIC approach to high performance network interface design: Protected DMA and other techniques," in *Proc. 16th Annu. Joint Conf. IEEE Comput. Commun. Soc. Driving Inf. Revol.*, Kobe, Japan, 1997, vol. 2, pp. 823–831.

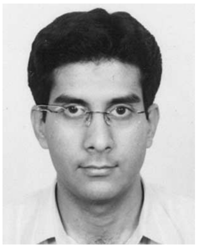
[18] S. Vitturi, L. Seno, F. Tramarin, and M. Bertocco, "On the rate adaptation techniques of IEEE 802.11 networks for industrial applications," *IEEE Trans. Ind. Informat.*, vol. 9, no. 1, pp. 198–208, Feb. 2013.

[19] G. Giorgi and C. Narduzzi, "Modeling and simulation analysis of PTP clock servo," in *Proc. IEEE Int. Symp. Precis. Clock Synch. Meas. Control Commun.*, Vienna, Austria, Oct. 2007, pp. 155–161.

[20] K. Salah and A. Qahtan, "Implementation and experimental performance evaluation of a hybrid interrupt-handling scheme," *Comput. Commun.*, vol. 32, no. 1, pp. 179–188, 2009.

[21] (2014). *Oregano Systems Syn1588 PCIe Network Interface Card* [Online]. Available: http://www.oreganosystems.at, accessed February 15, 2013.

[22] A. Mahmood and R. Exel, "Servo design for improved performance in software timestamping-assisted WLAN synchronization using IEEE 1588," in *Proc. Emerging Technol. Factory Autom.* (ETFA), Cagliari, Italy, Sep. 2013, pp. 1–8.



Aneeq Mahmood was born in 1982 in Lahore, Pakistan. He received the M.S. degree in electrical engineering from Kungliga Tekniska Högskolan (KTH), Stokholm, Sweden, in 2007, and the Ph.D. degree from Vienna University of Technology (VUT), Vienna, Austria, in 2013.

From 2007 to 2013, he was employed by the Austrian Academy of Sciences, Wien, Austria, where he worked on projects related to wireless localization and factory automation. Since June 2013, he has been working as a Project Assistant with the Institute of Computer Technology, VUT. His current research interests include wireless sensor networks, wired and wireless clock synchronization, WLANs, and energy optimization in automation.

Reinhard Exel (M'12) was born in Horn, Austria, in 1980. He received the Masters and the Ph.D. degrees in electrical engineering from the Vienna University of Technology, Vienna, Austria, in 2007 and 2012, respectively.

Since 2007, he has been with the Institute of Integrated Sensor Systems, Austrian Academy of Sciences, Wien, Austria, and with the Center for Integrated Sensor Systems, Danube University Krems, Austria, since 2013. His current research interests include accuracy improvement techniques for Ethernet-based clock synchronization systems, as well as multipath mitigation techniques for time-based real-time locating systems.

Thilo Sauter (M'93–SM'09–F'14) recieved the Diplom. Ing. and Ph.D. degrees in electrical engineering from the Vienna University of Technology (VUT), Vienna, Austria, in 1992 and 1999, respectively.

He worked with different institutes at VUT and became a Tenured Assistant Professor in 2006. Since 2004, he has also been the Director of the Institute for Integrated Sensor Systems, Austrian Academy of Sciences, Wien, Austria, which was transferred to the Danube University Krems, Krems, Austria, in 2013. His current research interests include IC design, smart sensors, and automation networks with a focus on real-time, security, interconnection, and integration issues.

Dr. Sauter is the President of the Austrian Association for Instrumentation and Automation, and an AdCom Member of the IEEE Industrial Electronics Society and the IEEE Sensors Council.