Hardware Efficient Clock Synchronization for Wi-Fi and Ethernet Based Multi-hop Network Using PTP

Muhammad Aslam, Wei Liu, Xianjun Jiao, Jetmir Haxhibeqiri, Gilson Miranda, Jeroen Hoebeke, Johann Marquez-Barja and Ingrid Moerman, *Member, IEEE*

***Abstract*—Precision Time Protocol (PTP), a state-of-the-art clock synchronization protocol primarily designed for wired networks, has recently gained attention in the wireless community, due to the increased use of IEEE 802.11 Wireless Local Area Networks (WLAN) in real time distributed systems. However, all the existing WLAN based PTP designs either incorporate software Timestamping (TS) delivering poor clock synchronization accuracy, or Hardware (HW) TS providing better synchronization accuracy at the cost of a significant amount of additional HW for TS support. Moreover, the performance of the existing PTP solutions is generally evaluated over simple single-hop networks, while the performance of these solutions over complex multi-hop networks is taken for granted. In this paper, a new Software Defined Radio (SDR) based approach to implement PTP is introduced and validated for IEEE 802.11 WLAN. Instead of using a dedicated HW clock, the prototype utilizes the Timing Synchronization Function (TSF) clock, which is already defined in IEEE802.11 standard for synchronization between access point and WLAN stations. The clock synchronization performance of our novel solution is thoroughly investigated over both single-hop WLAN and multi-hop wired-wireless networks. Experimental results unveil that 90% of the absolute clock synchronization error falls within 1.25 μs with our approach.**

***Index Terms*—PTP, IEEE 802.11, Wi-Fi, Ethernet, Clock Synchronization, Hardware Timestamping, openwifi, TSF, Wired-wireless Network.**

I. Introduction

CLOCK Synchronization (CS) is one of the prominent technologies for building real-time distributed networks. It enables the nodes in the distributed network to share the same notion of time. It is crucial for a system where performance highly depends on the CS accuracy of the networks. Examples of such systems include, Ultra-Reliable Low-Latency Communications (URLLC) in Industrial Wireless Sensor and Actuator Networks (IWSAN) [1], audio or video streaming over distributed networks [2], and network based motion control [3].

Precision Time Protocol (PTP) is a state-of-the-art CS protocol introduced in the IEEE 1588 standard [4]. It is the most frequently used CS protocol in wired distributed networks and is capable of providing sub-microsecond CS accuracy. The CS process in PTP is typically accomplished in two steps: establishing master-slave hierarchy, and synchronizing the master-slave clocks. In the former step, all the nodes in the network exchange announce messages containing information pertaining to their clocks' quality. The nodes later leverage the best master clock algorithm to elect a node with the best quality clock as a reference clock in the network. The reference clock acts as master and all the rest of the nodes' clocks are slaves. In the later step, the master clock periodically exchanges special messages with slave clocks and the type of these messages depends on the configured modes. While these special messages are *Sync, FollowUp, DelayReq*, and *DelayResp* in End-to-End (E2E) mode, the Peer-to-Peer (P2P) mode uses *Sync, FollowUp, PdelayReq, PdelayResp*, and *PDelayRespFollowUp* messages. Slave clocks in the network extract the master clock information from these messages and synchronize to it by computing the time difference. In principle, any of the two modes can be used, but they can also be combined in a network. Note that the PTP messages can be transported over different types of network layers such as PTP over UDP (encapsulated in IPv4 or IPv6 payload), and PTP over Layer 2 network (i.e., encapsulated in IEEE 802.3 payload without transport layer).

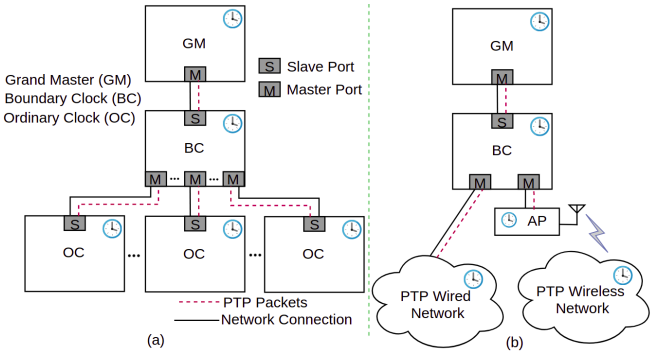

Fig. 1. An example of PTP based (a) conventional wired network and (b) wired-wireless hybrid network when configured in E2E mode.

The aforementioned process is not only applicable to a single hop network, but can be used to establish clock synchronization in a multi-hop network across different network domains (e.g., wired and wireless). A multi-hop PTP network consists of: (i) a Grand Master (GM) clock which is the primary clock for CS; (ii) one or more Boundary Clocks (BC) which can be configured as slave clock on one port and as master clock on the other ports; and (iii) single-port Ordinary Clock (OC) which can only be used either as slave or master

M. Aslam, W. Liu, X. Jiao, J. Haxhibeqiri, and J. Hoebeke, I. Moerman are with IMEC-IDLab, Department of Information Technology, Ghent University, Ghent, Belgium (e-mail: muhammad.aslam@ugent.be; wei.liu@ugent.be; xianjun.jiao@ugent.be; ingrid.moerman@ugent.be; jetmir.haxhibeqiri@ugent.be; jeroen.hoebeke@ugent.be).

G. Miranda and J. Marquez-Barja are with IMEC-IDLab, Antwerp University, Antwerp, Belgium (e-mail: gilson.miranda@uantwerpen.be; johann.marquez-barja@uantwerpen.be).

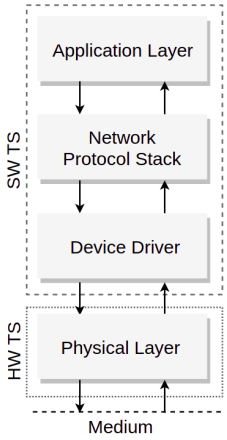

Fig. 2. Locations for timestamping in clock synchronization protocols, where HW TS and SW TS denote hardware timestamping and software timestamping, respectively.

clock. An example of a practical multi-hop PTP based CS network configured in E2E mode is shown in Fig. 1-a. Alternatively, BC can be replaced by Transparent Clock (TC), BC and TC are mathematically equivalent in terms of time offset calculation for CS. The main difference is that TC forwards PTP messages to the next hop and compensates the time the messages reside in the device (residence time), whereas BC generates new PTP messages. For practical reasons, in this paper we use BC in the E2E mode.

The CS accuracy of PTP is defined as the remaining time difference between a master and a slave which cannot be further reduced. The CS accuracy of PTP depends upon the timestamps captured at the moment of reception or transmission of packets, among others. Fig. 2 shows the different locations for Timestamping (TS) in PTP. TS in PTP can be achieved via Hardware (HW) or Software (SW). HW TS is produced at or relatively close to the physical layer and it is realized by using dedicated HW to assist the TS. On the other hand, SW TS is generated in device drivers or at a higher layer of the network stack without using any dedicated HW. The CS accuracy is significantly affected by the location of TS, and higher CS accuracy requires the TS location to be as close as possible to the physical layer, so that the timestamp is minimally affected by the time variation caused by packets travelling through the network stack. Since HW TS happens at or close to the physical layer, as illustrated in Fig. 2, a HW TS based PTP clock is generally more accurate than a SW TS based PTP clock.

Most of the PTP based networks have been implemented over the wire [5]–[8]. For instance, CIPSync is a PTP compliant industrial solution which provides CS accuracy of a few nanoseconds over conventional Ethernet [7]. Another example is PROFINET, a standard for data communication over industrial Ethernet, which uses PTP as its CS protocol [8]. The industry however is increasingly interested in extending PTP from wired network to wireless in the form of a hybrid multi-hop network for more flexibility, increased scalability and reduced deployment cost [9]. An example of a simple hybrid multi-hop is shown in Fig. 1-b. This has led to the introduction of Timing Measurement (TM) and subsequently Fine Timing Measurement (FTM), which are the extensions of IEEE 802.1AS standard to enable CS over IEEE802.11 network [10].

Academic researchers have proposed several wireless CS solutions [11]–[17]. SW TS based PTP solutions are simple and easy to realize, but they provide relatively poor CS accuracy in the range of several microseconds [11]–[13]. The CS accuracy can be further degraded due to propagation delay asymmetry potentially produced from different modulation and coding schemes (MCS) [13]. On the other hand, HW TS based PTP performs better, but, at the cost of dedicated HW blocks added to realize HW TS [14]–[17]. Moreover, the CS performance of these PTP solutions is mostly analysed on a single-hop network. The performance of these solutions over complex multi-hop network is either taken for granted or simply left as future work. In addition to academic research, a few industrial products are known on this topic [18], [19]. However, upon further exploration, it is found that these solutions, which claim to support FTM, can only be used for ranging applications, as the FTM TS is not exposed to the Linux Operating System (OS). Thus, to the best of our knowledge, these solutions cannot be directly used for CS applications.

In this paper, we have made an effort to enable HW TS based PTP in *openwifi* [20], an open-source chip design of IEEE 802.11. Unlike previous work which uses an additional clock for HW TS, we have leveraged the existing Timing Synchronization Function (TSF) clock with minimal modifications of *openwifi* in Field Programmable Gate Array (FPGA) to achieve the same purpose. The approach is not only hardware efficient, but also allows the existing TSF based CS protocols to improve CS accuracy (e.g., Time Advertisement (TA) mechanism introduced in IEEE 802.11 standard). Moreover, we add necessary callbacks in the *openwifi* driver to make the TSF clock compliant to the PTP Hardware Clock (PHC) subsystem of Linux. In this way, we can use the existing PTP application and the support infrastructure in Linux [21]. Lastly, the performance of our solution is characterized over both single-hop and a multi-hop network across wired and wireless network domains.

The rest of the paper is structured as follows. Related work is detailed in Section II. Section III describes our method to enable PTP on IEEE 802.11 network. Experimental results are discussed in Section IV. Lastly, conclusions and future work are given in Section V.

II. State of the Art

There have been many attempts to implement PTP over IEEE 802.11 Wireless Local Area Network (WLAN). These works can be categorized into SW TS based PTP, and HW TS based PTP implementations.

It is worth noting that the metrics commonly used in literature to evaluate the performance of these solutions are the mean (μ) and standard deviation (σ) of CS error over time. Given C_{err} represents CS error between the reference clock (i.e., master clock) and the clock to be synchronized

(i.e., slave clock), these values are calculated by Eq. 1, Eq. 2, respectively.

$$\mu = \frac{1}{N}\sum_{n=1}^{N} C_{err(n)} \tag{1}$$

$$\sigma = \sqrt{\frac{1}{N-1}\sum_{n=1}^{N}(C_{err(n)} - \mu)^2} \tag{2}$$

where N represents the total number of available CS error samples. In addition to these metrics, the Wi-Fi Alliance has announced a *Wi-Fi TimeSync* certificate to specify the requirement of CS performance between multiple Wi-Fi devices [22]. The certification requires the 90th percentile of the absolute CS error (P_{90}) to be within 5.5 μs for 90% of the observed time (i.e., 120 sec). In this paper, Eq. 3 is used to quantify the P_{90}

$$P_{90} = \begin{cases} i, & \text{if } f(i) = 0.9 \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

$$f(i) = \frac{1}{N}\sum_{n=1}^{N} 1_{C_{err(n)}} \quad where$$
$$1_{C_{err(n)}} = \begin{cases} 1, & \text{if } |C_{err}(n)| \leq i \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

where $f(i)$ (empirical cumulative distribution function) represents the percentile of absolute CS error at ith point. In other words, the i value denotes the P_{90} when $f(i)$ is equal to 0.9. To check the compliance with *Wi-Fi TimeSync* certificate, we include P_{90} together with the mean and standard metrics to measure the CS performance of our solution in a network.

A. PTP with software timestamping

SW TS generally happens at the application layer, within the network protocol stack, or in the device driver of a radio (see Fig. 2). Chen Wu et al. [11] have proposed a PTP solution over IEEE 802.11 WLAN with SW TS realized at the application layer. To mitigate the impact of asymmetric bidirectional delay of WLAN on the PTP performance, they have first designed a delay filtering algorithm based on Kalman filter, and then introduced a modified Proportional Integral (PI) controller based clock servo system. The proposed solution is validated on a Linux based embedded development board. The CS accuracy is limited to -14.24 μs with a standard deviation of 27.65 μs. Authors in [12] have investigated a PTP solution over IEEE 802.11 WLAN, where SW TS is done inside the interrupt handling routine of the Ath5k[1] driver. The Ath5k is a radio driver used by Atheros AR5xxx chipset family in the Linux kernel. However, the mean CS accuracy of this work is still in the range of several microseconds; i.e, 6.60 μs with 0.58 μs standard deviation [13].

Another design of PTP over IEEE 802.11b WLAN is investigated in a Linux Personal Computer (PC) [16]. To improve the SW TS, they have made changes in the radio driver of Linux. The approach has brought better CS accuracy (i.e., average offset error of 4.6 μs with 1.58 μs standard deviation) when configured in Access Point (AP) mode.

The major missing pieces of the aforementioned solutions are that: (i) the CS performance is only analyzed over single-hop wireless network, and (ii) the CS performance is evaluated with PTP packets exclusively (i.e., without any other traffic load on the network besides PTP messages). An ambitious effort is made in [23] to investigate the performance of PTP over multi-hop hybrid network. The work has however used SW TS in the wireless part of the network and the performance of their work is not tested in the presence of non-PTP background traffic.

To summarize, all these SW TS based solutions have been realized over commercial off-the-shelf WLAN chipsets without any hardware modification. They provide decent CS accuracy (in the order of several microsecond). However the CS accuracy is evaluated in rather simple scenarios (e.g., no traffic load). On top of that, the propagation delay asymmetry potentially generated from different frame sizes and MCS can significantly harm CS performance of the solutions [13], [24]. Although these solutions provide important insights for PTP implementation in wireless network, we believe they are not adequate in terms of stability for industrial applications [25].

B. PTP with hardware timestamping

HW TS is realized at or close to the physical layer of a radio (see Fig. 2). The approach enables PTP to achieve better CS performance than SW TS based PTP in general over WLAN. A HW TS at the physical layer for a modified PTP has been prototyped over IEEE 802.11b WLAN [17]. In order to achieve TS, the work has implemented a dedicated Adder Based Clock (ABC) in FPGA. In the prototype, a customized version of the PTP protocol is used in which, instead of using *Sync* and *FollowUp* messages, the synchronization data from master to slave is embedded within beacons. Experimental results show that the solution has the virtues of high CS accuracy; i.e., mean CS accuracy is 0.24 ns with standard deviation of 0.53 ns. We believe apart from the apparent benefit of HW TS, the solution's high CS accuracy is also thanks to the frequent beacon broadcast (i.e. typically 10 times a second). The results however do not incorporate any information regarding the impact of traffic load on the CS accuracy nor its performance over multi-hop networks. Another HW TS based PTP solution is analyzed over WLAN by using an embedded processor and Programmable Logic Device (PLD) in [16]. To realize HW TS, they have implemented 4 hardware modules including two 64 bits counters on the PLD. The solution gives a mean CS accuracy of 1.1 ns with a standard deviation of 1.76 ns, but it uses a custom Media Access Control (MAC) mechanism, which is not compatible with the IEEE 802.11 standard. The CS performance of all these HW TS based solutions is quantified over a single-hop wireless network and none of them have mentioned their solution's performance when non-PTP traffic is present in the network. Further, all these works have utilized a dedicated HW clock to realize TS. In other words, an additional HW clock is required to enable PTP in commercial products, making

[1]https://wireless.wiki.kernel.org/en/users/Drivers/ath5k

them less economical and no more compliant with the 802.11 standard.

C. Contribution of this paper

The key characteristics and benefits of the PTP solution presented in this work are summarized as follows:

1) The presented PTP solution supports HW assisted TS, making it more accurate than SW TS based PTP in terms of CS accuracy.
2) Unlike the previous HW TS based PTP solutions which use additional HW clocks for HW TS, this work is more economical in terms of hardware footprint, as it uses the existing TSF clock for HW TS.
3) Existing TSF based solutions [26], [27] only perform clock offset correction during the CS process. The CS accuracy however requires both clock offset and skew correction [28]. The CS process of the HW clock to be synchronized (C_s) with the master clock (C_m) can be modeled by Eq. 5

$$C_s(t) = S \times C_m(t) + b \tag{5}$$

where S and b represent skew and offset of HW clock [29]. Clock offset is the relative time difference between master and slave, whereas clock skew is the relative difference in clock frequency between the master and slave clocks. It implies that the clock skew between master and slave remains uncorrected in the existing TSF based solutions. As a result, the clocks quickly diverge after each correction. IEEE 802.11 [30] has specified ± 20 ppm frequency skew for WLAN chipsets, resulting in CS errors of up to ± 40 μs per sec if only clock offset correction is applied once per second. Thus, we also introduce skew correction capability in conjunction with clock offset correction in a TSF clock by slightly modifying the existing TSF block of *openwifi* in FPGA.
4) Our solution has the potential to further enhance the CS accuracy of the existing TSF based CS protocols (e.g., TA mechanism in IEEE 802.11 standard), due to the introduction of clock skew correction in TSF clock.
5) Our solution is based on an open source radio chip design, making it more convenient for users to adopt the approach.
6) Our solution is full stack and Linux compatible, using existing PTP software and Linux kernel support.
7) Unlike existing work, we have quantified the performance of our prototype in 90 percentile of absolute CS error to validate the compliance of our solution with the *Wi-Fi TimeSync* certificate.
8) Lastly, to examine the adequacy of our solution for industrial applications, the CS performance of the solution is investigated over a single-hop wireless network as well as a multi-hop hybrid network, coexisting with a large amount of non-PTP traffic.

III. The Proposed Solution

Fig. 3 represents the block diagram of our proposed architecture for PTP design over IEEE 802.11 WLAN. The architecture is prototyped on a System on Chip (SoC) where the *openwifi*, an open-source chip design of IEEE 802.11 WLAN, is running. Additionally, PTP software stack and PTP HW clock are realized on embedded processor and FPGA parts of the SoC, respectively. To implement the PTP software stack, we have employed the existing *linuxptp* software [31] as a userspace application. *Linuxptp* software, which is the Linux based PTP implementation for wired (Ethernet) networks, relies on certain system calls to determine a device's TS capability (i.e., SW or HW TS). These calls are unfortunately not accessible for a Wi-Fi card driver without Linux kernel modifications, hence changes are made in the application to bypass the check without hurting the *Linuxptp* software's compliance with PTP standard. We have further modified the radio device driver, making it able to interface with the PTP HW clock. Lastly, the necessary changes are made in the existing TSF hardware to use it as the PTP HW clock. These changes enable TSF to perform both clock offset and skew corrections, where clock offset means the time difference between master and slave, and clock skew is the frequency difference between the master and slave's clocks. In short, the proposed PTP architecture can be generalised into two crucial extensions: (1) design a wireless PTP software stack, and (2) modification of the existing TSF clock in HW to allow PTP HW TS. These two extensions are detailed in this section.

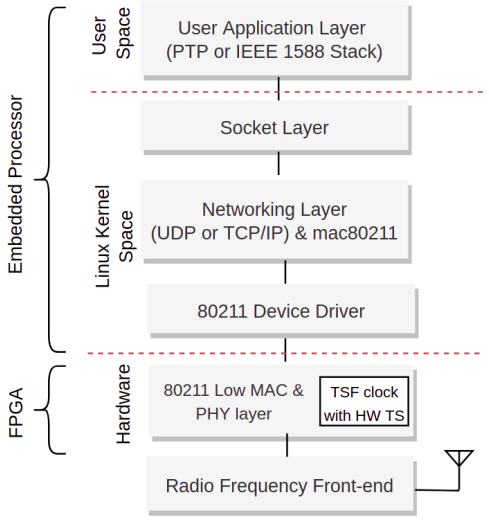

Fig. 3. Block diagram of our proposed PTP design over WLAN.

A. Design of the PTP software stack

A generalized diagram containing the main components used in the PTP software stack is shown in Fig. 4. There is an existing PTP clock infrastructure, also referred to as the PTP Hardware Clock (PHC) subsystem, in the kernel of Linux OS. The PHC subsystem offers Application Programming Interfaces (APIs) for both userspace applications as well as for device drivers to control the HW clock. Introducing PTP clock support for an WLAN interface requires the integration of these APIs together with TS at both userspace level and device driver.

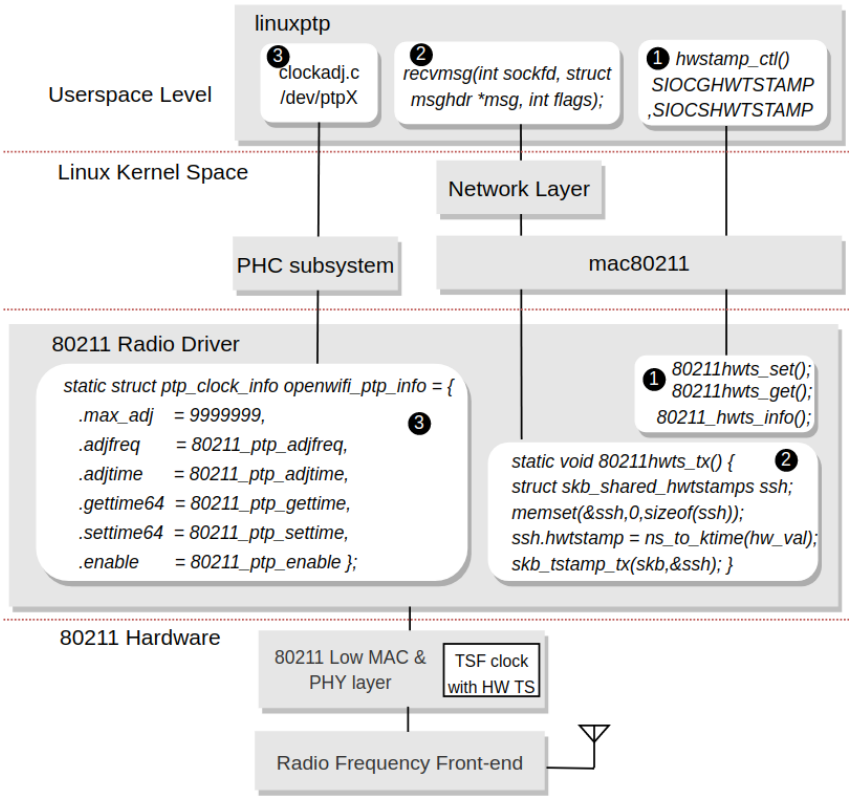

Fig. 4. A generalized diagram containing the main components used in the PTP software stack.

In the design, we have used *linuxptp* as the software stack of PTP clock at userspace level. *Linuxptp* uses the APIs of the PHC subsystem in combination with *SO_TIMESTAMPING* socket option to regulate the HW clock. *SO_TIMESTAMPING* is a socket attribute used by *recvmsg()* at userspace level to generate timestamps on transmission, reception or both. The socket attribute supports both HW and SW TS. To run *linuxptp* over WLAN, it is essential that the radio driver has support for the PHC interface as well as *SO_TIMESTAMPING* socket option. To this end, the following steps are taken to run *linuxptp* over *openwifi* in the radio driver:

1) *linuxptp* uses *SIOCSHWTSTAMP* in an ioctl system call to configure which outgoing and incoming packets should be timestamped. The selection of the packets is necessary, because not all the packets on the network stack require HW TS. For instance, there is no need for TS in non-PTP packets. The corresponding callback function implemented in our driver is *80211hwts_set* (see point 1 in Fig. 4). *SIOCGHWTSTAMP* is an ioctl system call used to read the already configured packet settings and its callback function in the driver is *80211hwts_get*. *80211hwts_info* is another callback function which returns the TS capabilities of a Network Interface Card (NIC), when *ethtool*[2] *-T sdr0* command is executed at userspace level, where *sdr0* represents the Wi-Fi interface formed by openwifi as recognized by the Linux OS.
2) After configuring the HW timestamp settings for PTP packets, the next step is to enable the HW TS in the radio driver. *sk_buff* is a data structure in the Linux networking stack designed to handle the packet that has been received or is about to be transmitted. The structure allows the HW timestamps to be stored in the field *skb_shared_hwtstamps* optionally. A device driver is responsible for reading the timestamps from hardware registers and writing them into the *skb_shared_hwtstamps*. The function callback where the radio driver copies TS value from HW register to *sk_buff* upon packet transmission is shown in Fig. 4 (see the point 2 at the radio drivel level).
3) The last step is to expose the HW clock to *linuxptp*, so that it can regulate the clock from the userspace level. The *ptp_clock* is a structure representing the PTP clock in a radio driver. It provides an abstraction on top of the HW clock and allows the userspace application to get, set and adjust the HW clock automatically. An example of the code snippet with the key functionalities of *ptp_clock* is shown in Fig. 4 (see point 3 at the radio driver level).
   The *80211_ptp_adjtime* and *80211_ptp_adjfreq* are the most important callback functions, among others. The former function performs clock offset correction and the latter does skew correction of PTP HW clock. The clock skew correction value is calculated by using Eq. 6.

[2]ethtool command: https://linux.die.net/man/8/ethtool

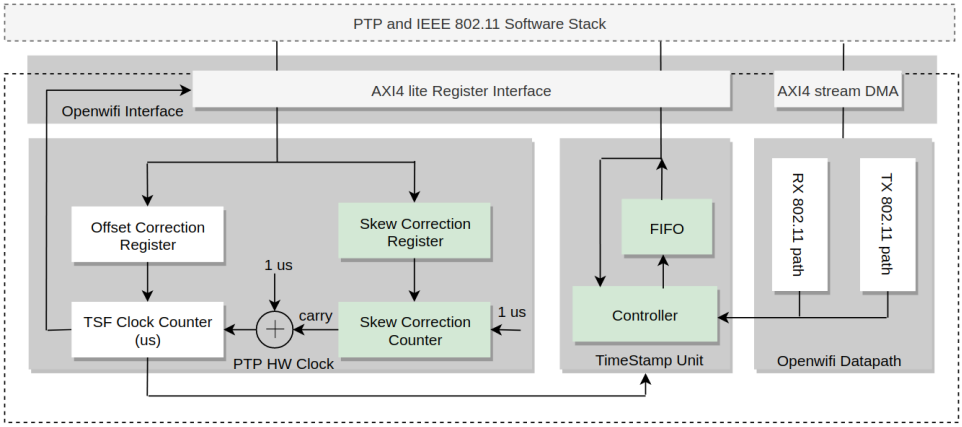

Fig. 5. A diagram of the hardware components added (highlighted in green) to assist with hardware timestamping.

$$S_c = \left(\frac{10^9}{F_r \times ppb}\right) \times 10^6 \tag{6}$$

Where, *linuxptp* computes the ppb (part per billion) value based on the received HW timestamps, and sends it to the radio driver. Our radio driver uses this value to calculate the S_c (skew correction) value, which corresponds to the duration in microseconds that the clock is adjusted periodically based on the ppb value. F_r denotes the desired frequency in Hz of PTP HW clock. Let's say the quantified ppb value is 78,096 and F_r value is 1 MHz, then the actual frequency of PTP HW clock becomes 1.000078096 MHz. The calculated S_c value by using Eq. 6 is 12,805 μs, which is later used by PTP HW clock (see Fig.5) to correct clock skew (further explained in the next section). Lastly, the ptp_clock is exposed as a character device ($/dev/ptpX$) to userspace, where *linuxptp* directly uses it to discipline the HW clock.

B. Design the assisting HW for PTP HW TS

As shown in Fig. 5, the HW unit used in our design primarily consists of (1) PTP HW clock representing real time local HW clock, (2) TimeStamp Unit (TSU) responsible for generating HW TS upon detection of reception or transmission of frame preamble, and (3) IEEE 802.11 datapath employed for sending and receiving PTP packets.

There are two types of interfaces used on the SoC to transfer data between the processor and the openwifi hardware, both belong to the 4th revision of Advanced eXtensible Interface (AXI4) [32]. One type is used for register configurations (i.e. AXI4 Lite); whereas the other is for high speed and high volume data transfer (i.e., AXI4 Stream), being the interactions with Direct Memory Access (DMA) for transmitting and receiving packets in the 802.11 datapath of openwifi. More specifically for PTP realization, the relevant registers (e.g., Offset Correction Register, Skew Correction Register) are configured with AXI4 Lite interface, while the PTP messages generated by the PTP software stack are sent/received via AXI4 Stream interface.

Instead of utilizing a dedicated clock, PTP HW clock is realized by leveraging the existing TSF clock. While HW CS demands both clock offset and skew correction, the default TSF implementation is only able to do offset correction. Thus, we have modified the TSF HW making it capable of performing skew correction as well (see colored blocks in Fig. 5). The radio driver calculates the upper limit (i.e. the S_c value) for Skew Correction Counter (SCC) using Eq. 6, and loads the calculated value into the Skew Correction Register (SCR) via AXI4 Lite interface [32]. SCC increments with the TSF clock at 1MHz operating frequency. Upon reaching the S_c value, SCC overflows and performs the skew correction of the TSF clock. In normal situations, the TSF counter is incremented by one after each 1 μs according to the desired rate of the local oscillator. Upon skew correction, the TSF counter is either incremented by two or zero depending upon the sign of the carry bit. For instance, SCC performs skew correction after each 12,805 counts in the example given in the previous section.

Lastly, the TSU is composed of a controller and a First-In, First-Out (FIFO). The controller is in charge of executing TS upon either frame preamble transmission (or reception), and storing it into the FIFO. The radio driver later reads and copies these TS values into *sk_buff* by calling *80211hwts_tx()* function (see point 2 at the radio drivel level in Fig. 5). The radio driver also performs a similar action for the reception process. For simplicity, this is not shown in Fig. 5. The radio driver performs all these actions inside the Interrupt Service Routine (ISR). Subsequently, the *linuxptp* calls *recvmsg()* function callback to read these TS values and performs CS.

IV. Results and Discussions

In this section, first, the experimental setup used to evaluate the CS performance of our design is presented. Then, the CS performance of our design is analysed over both a single-hop network and multi-hop network across wired and wireless network segments. The CS performance of our design is quantified in terms of CS mean value (μ), standard deviation (σ) and the 90th percentile of the absolute CS error (P_{90}). These values are calculated by Eq. 1, Eq. 2, and Eq. 3, respectively. Lastly, the CS performance of our

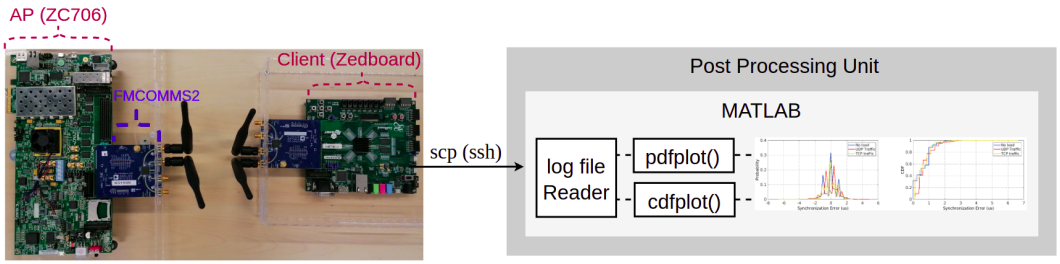

Fig. 6. Experimental setup of the single-hop network used to measure the performance of our proposed PTP over WLAN.

design is compared against the performance of existing CS solutions. Since the existing solutions in literature only provide information regarding μ and σ values, we have estimated P_{90} value with the help of Eq. 7. We assume that the samples in a CS error measurement follow a Gaussian distribution, which implies that its absolute values will follow a folded Gaussian distribution, whose cumulative distribution function is given in Eq. 7.

$$f(C_{err}) = \frac{1}{2}\left[erf\left(\frac{C_{err}+\mu}{\sigma\sqrt{2}}\right) + erf\left(\frac{C_{err}-\mu}{\sigma\sqrt{2}}\right)\right] \quad (7)$$

$$erf\left(\frac{C_{err}\pm\mu}{\sigma\sqrt{2}}\right) = \frac{2}{\sqrt{\pi}}\int_0^{\left(\frac{C_{err}\pm\mu}{\sigma\sqrt{2}}\right)} e^{-t^2}\,dt \quad (8)$$

where $f(C_{err})$ reflects the estimated percentile of the absolute CS error at $|C_{err}|$th point. In other words, the $|C_{err}|$ value corresponds to the estimated P_{90} when $f(C_{err})$ is equal to 0.9.

A. Experimental Setup for a Single-Hop Network

To measure the performance of our proposed PTP solution with HW TS over single-hop WLAN, a wireless network between two *openwifi* Software Defined Radio (SDR) boards has been established, as illustrated in Fig. 6. The WLAN is set up in infrastructure mode, where one of the SDRs acts as the Access Point (AP) and the other behaves as a client. Due to the lack of power amplifier in the used RF frontend, the SDRs are placed in close proximity (i.e., the measured transmit power of the used SDR is -15 dBm). The SDR used as the AP in the particular experiment is composed of ZC706 [33], and FMCOMMS2 [34], an analog RF frontend. The SDR used as the client consists of Zedboard [35], a development board for Zynq 7000 SoC [36], and FMCOMMS2. Both ZC706 and Zedboard are using Xilinx Zynq 7000 SoC, the main difference is the FPGA size and the peripherals on the boards. We do choose two different types of FPGA boards intentionally, as oscillators on different boards tend to have bigger differences, hence the clocks based on different boards are more challenging to synchronize. The Zynq 7000 SoC further comprises Programmable Logic (PL) (FPGA) and Processing Subsystem (PS) (ARM Cortex-A9). The PTP hardware unit along with the low MAC and physical layers of *openwifi* are implemented in the PL part, while high MAC and other layers of network stacks of *openwifi* [37] and PTP software stack are running on the embedded PS part. The operating system running on the PS is Linux with kernel version 4.14. In the particular setup, the *openwifi* is configured in IEEE 802.11a mode operating in 5 GHz frequency band with 20MHz channel bandwidth. The Modulation and Coding Scheme (MCS) values for IEEE 802.11a are dynamically adapted up to 7 by Linux *mac80211* minstrel rate control algorithm.

TABLE I
THE MEASURED CLOCK SYNCHRONIZATION PERFORMANCE OF OUR PROPOSED PTP OVER SINGLE-HOP WLAN.

Parameters	No load	TCP load	UDP load
μ	-0.018 μs	0.007 μs	0.006 μs
σ	0.840 μs	0.824 μs	0.919 μs
P_{90}	1.25 μs	1.36 μs	1.38 μs

The AP acts as the PTP master and the client is the PTP slave. In our setup, PTP is configured in E2E mode. Thus, the master and slave periodically exchange *Sync, FollowUp, DelayReq*, and *DelayResp* messages during the CS process. These messages in the *linuxptp* are transmitted or received using UDP/IP via sockets API (see Fig. 3). Note that the PTP messages in our setup are transported over UDP and IPv4. The CS accuracy of our prototype is measured by a software report generated in *linuxptp*. The report includes master offset (i.e., the measured CS error in nanosecond), frequency offset (i.e., the measured clock skew between the PTP master and slave clocks in *ppb*). The *linuxptp* v2.0 is used in the experiment and is configured to give updates on these metrics once per second. The report is fed to the post processing unit indicated in Fig. 6, where master offset and frequency offset samples are extracted from the report and the CS performance metrics (i.e. μ, σ and P_{90}) are derived.

B. Experimental Evaluation over single-hop network

Before evaluating the CS accuracy over the single-hop WLAN, we first evaluate the Convergence Time (CT) of our solution. CT is the time acquired by PTP to reach its stable phase. Short CT is always desired, since time critical applications can run only when CS error is stable and small enough.

1) Convergence Time: We have quantified the CT of our proposed PTP solution in terms of CS error and frequency offset under two different experimental setups. In the first setup, we have used different SDR boards (i.e., ZC706 board acts as master and Zedboard functions as slave as depicted in Fig. 6). In the second setup, ZC706 board is replaced with another Zedboard. The main reason behind using different

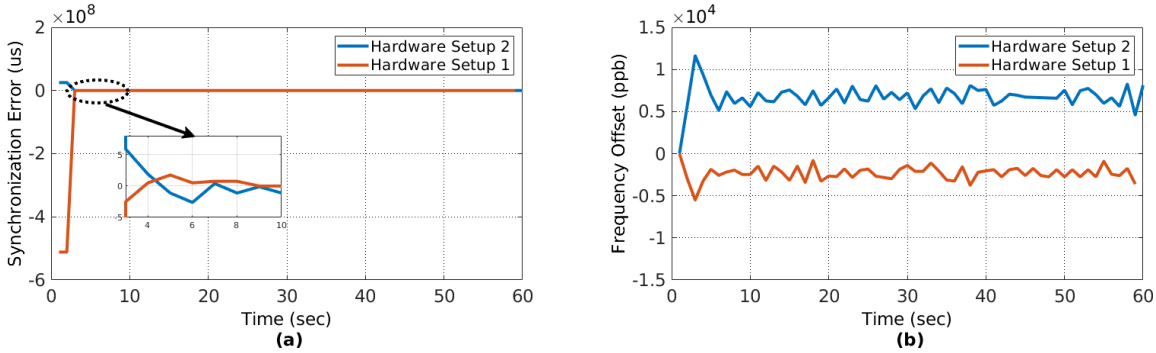

Fig. 7. Convergence time of the proposed PTP solution over WLAN in terms of (a) clock synchronization error and (b) frequency offset.

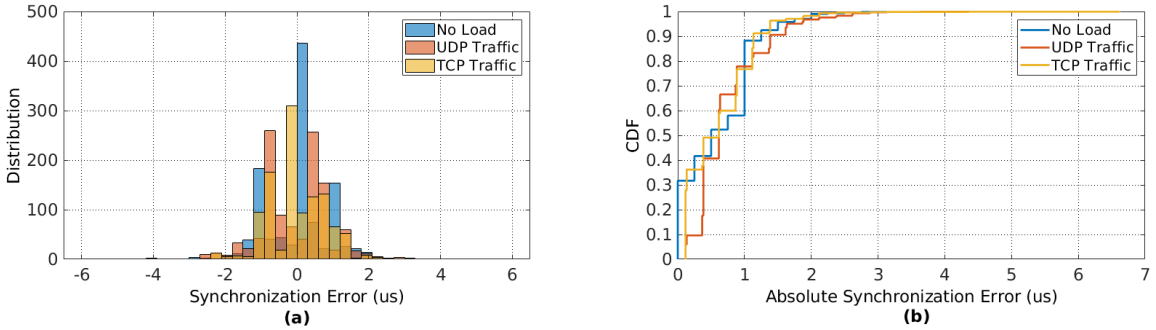

Fig. 8. (a) Histogram of the clock synchronization error, and (b) the Cumulative Distribution Function (CDF) of the absolute clock synchronization error of our proposed PTP solution over single-hop WLAN when no traffic load, UDP traffic and TCP traffic is applied.

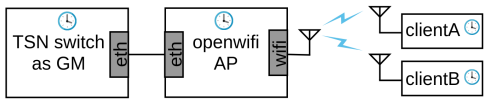

Fig. 9. Experimental setup used to measure the performance of our proposed PTP over multi-hop hybrid network.

HW setups is to verify whether the CT performance of our proposed PTP is affected by the HW formations.

Fig. 7 displays the CT of our proposed PTP solution over WLAN under the two different HW setups. We have defined CT as the time when CS error or frequency offset reaches a stable range. The stable range can be positive or negative depending on whether the slave clock is leading or lagging behind the master clock. Fig. 7-a illustrates the CT in terms of CS error. It can be observed from Fig. 7-a that the two curves of CS error for two different HW setups converge to almost the same stable range after approximately 4 sec (i.e., CT value). The stable range of CS error for the first setup (see hardware setup 1 in Fig. 7-a) is ± 0.81 μs with a μ value of 0.071 μs. Similarly, the stable range of CS error for the second setup (see hardware setup 2 in Fig. 7-a) is ± 1.03 μs with a μ value of -0.04 μs. To highlight the transition to the stable phase of PTP, a zoomed-in trace is also shown in Fig. 7-a.

Similarly, the report generated by *linuxptp* also contains the frequency offset of the slave clock relative to master. The frequency offset can also be used to compute the CT of PTP, as displayed in Fig. 7-b. Unlike the convergence curves of CS error (see Fig. 7-a), the curves of frequency offset for the two different HW setups never converge to the same stable range. This is because different HWs have crystal clocks with slightly different frequency error. The stable range of frequency offset for the first setup (see hardware setup 1 in Fig. 7-b) is ± 675 *ppb* with a mean value of -2309 *ppb*. Similarly, the stable range of the frequency offset for the second setup (see hardware setup 2 in Fig. 7-b) is ± 904 *ppb* with a mean value of 6809 *ppb*. Though the convergence curve of frequency offset depends on the HW crystal clock, it gets stabilized during the stable phase of PTP. As shown in Fig. 7-b, the CT value measured in terms of frequency offset is approximately 6 sec. Apparently, this value is higher than the CT value estimated based on the CS error. During the start up process, *linuxptp* sequentially travels through three different clock servo states: unlocked (s0), clock-step (s1), and locked (s2). The *linuxptp* performs only clock offset correction during s1 state. The frequency offset correction begins when it enters into s2 state. Due to this, CT value when estimated based on frequency offset is relatively high. Thus, we can draw a conclusion from the experimental results that the CT of the proposed PTP can be measured by using CS error or frequency offset and it takes approximately 4 to 6 master-slave interactions on the typical SDR platform with crystal oscillator frequency error of ± 50 *ppm* [33] to get stablized. The CT value can be enhanced by shortening the interaction interval between master-slave in a wireless network.

2) Clock Synchronization Accuracy: A typical real time wireless network in an industrial environment contains back-

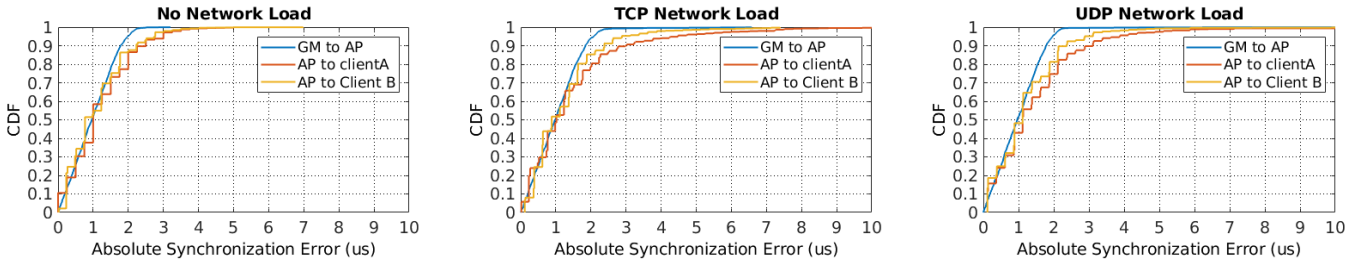

Fig. 10. Cumulative Distribution Function (CDF) of absolute clock synchronization error of our proposed PTP over multi-hop hybrid network when no traffic load, UDP traffic and TCP traffic is applied.

TABLE II
THE MEASURED CLOCK SYNCHRONIZATION PERFORMANCE OF OUR PROPOSED PTP WITHIN THE MULTI-HOP HYBRID NETWORK.

Description	No load			TCP load			UDP load		
	μ (μs)	σ (μs)	P_{90} (μs)	μ (μs)	σ (μs)	P_{90} (μs)	μ (μs)	σ (μs)	P_{90} (μs)
AP to GM	0.001	1.151	1.78	0.011	1.233	1.85	0.001	1.310	1.79
Client A to AP	0.003	1.492	2.23	0.002	2.249	2.38	-0.002	2.063	2.37
Client B to AP	0.002	1.375	2.50	-0.004	1.606	2.98	0.003	1.612	3.12
Client A to GM	0.004	2.643	4.01	0.013	3.482	4.23	-0.001	3.373	4.16
Client B to GM	0.003	2.526	4.28	0.007	2.839	4.83	0.004	2.922	4.91

ground network load in addition to PTP messages which can adversely affect the CS performance. For instance, network load could cause network congestion resulting in delays or drops of PTP packets. To this end, the PTP CS performance is quantified when no network load is present between the SDRs to show the optimum performance, and also when network load is present between the SDRs to show the performance under more practical conditions. Note that we have measured μ, σ and P_{90} of CS performance by using Eq. 1, Eq. 2, and Eq. 3, respectively, for single-hop network.

Network load defines how much other traffic together with PTP messages is transferred over the WLAN. The HW setup displayed in Fig. 6 is leveraged to quantify the CS error as we have already learnt from the previous section that the error is HW independent. Note that the experimental evaluation uses results of the stable phase of PTP. In other words, reports generated during the CT of PTP are ignored. The measurements last for 20 minutes with the CS error stamped once per second.

The CS errors quantified under different network loads are shown in Fig. 8 and Table. I. As seen from Fig. 8, the P_{90} is 1.25 μs with σ of 0.84 μs when no traffic load is applied. For network load settings, the iperf[3] software is used to generate traffic between the SDRs. First a TCP iperf stream and then a UDP stream is enabled from the client under test to the AP in conjunction with PTP message exchanges. Since TCP is by design bi-directional, i.e. using TCP of iperf will automatically find the maximum throughput (i.e. 16.4 Mb/s in this experiment) between the client and AP, it shows that we can push the limit of traffic load between the SDRs while keeping PTP running stable. The P_{90} rises to 1.36 μs with σ of 0.82 μs while running the TCP traffic (see Fig. 8). Lastly, as UDP is by design unidirectional, we have applied the average throughput found in TCP as the target throughput in UDP stream. The measurement shows that the design can also run with unidirectional traffic load without any feedback for flow control. The P_{90} jumps to 1.38 μs with σ of 0.92 μs when 16.4 Mb/s UDP iperf is enabled. It can be seen from the results that the impact of network load on the CS error is almost negligible; i.e., the P_{90} only increases by 0.13 μs and 0.11 μs with UDP and TCP traffic load applied, respectively, when compared against the case when no load is applied. Last but not least, experimental results have verified that our solution is capable of providing a P_{90} much better than 5.5 μs, which is required for *Wi-Fi TimeSync* [22] certification, in both optimum and more practical scenarios.

C. Experimental evaluation over multi-hop hybrid network

The experimental setup used to measure the CS performance of our solution over a multi-hop hybrid network is deployed in the w-iLab.t testbed [38]. Fig. 9 displays the high level overview of this setup. A Time Sensitive Network (TSN) capable switch [39] is configured as PTP GM. The *openwifi* AP composed of Zynq UltraScale+ MPSoC ZCU102 Evaluation Kit [40] and FMCOMMS2 behaves as a BC. The AP has two PTP HW clocks. The HW clock attached to Ethernet interface acts as a slave clock to the TSN switch and the HW clock attached to the *openwifi* interface functions as the master clock to the wireless network segment. Within the AP, the openwifi interface's PTP HW clock is synchronized against the Ethernet's PTP HW clock, using *phc2sys* command offered in the *linuxptp* software. The wireless network in the multi-hop hybrid network includes two Wi-Fi clients. Each *openwifi* client is made up of a ZC706 and an FMCOMMS2 board. The HW clocks of the two clients in the wireless network are slaves to the AP's openwifi PTP HW clock. The distance between each client and AP is approximately 3 meters, determined by the available mounting devices in the testbed. Since the RF frontend used in AP and clients lacks built-in power amplifier, an external power amplifier is connected at the transmitter

[3]Iperf https://iperf.fr/

TABLE III
PERFORMANCE COMPARISON WITH EXISTING CLOCK SYNCHRONIZATION SOLUTIONS OF WLAN IN THE LITERATURE.

	HW TS	Extra HW clock	Sync interval (sec)	Clock synchronization performance over single-hop network						Clock synchronization performance over multi-hop wired-wireless hybrid network					
				No load			Network load			No load			Network load		
				μ	σ	P_{90}	μ	σ	P_{90}	μ	σ	P_{90}	μ	σ	P_{90}
PTP based clock synchronization solutions															
Our	√	×	1.0	-018 ns	840 ns	1.25 μs	6.00 ns	919 ns	1.38 μs	4.0 ns	2.64 μs	4.01 μs	-1.0 ns	3.4 μs	4.2 μs
[17]	√	√	0.1	0.24 ns	0.53 ns	**1.00** ns	NA	NA	NA	NA	NA	NA	NA	NA	NA
[16]	√	√	2.0	1.10 ns	1.76 ns	**3.40** ns	NA	NA	NA	NA	NA	NA	NA	NA	NA
[11]	×	×	2.0	-14.2 μs	27.7 μs	**51.2** μs	NA	NA	NA	NA	NA	NA	NA	NA	NA
[12]	×	×	1.0	6.60 μs	0.58 μs	**7.34** μs	NA	NA	NA	NA	NA	NA	NA	NA	NA
[16]	×	×	2.0	4.60 μs	1.58 μs	**6.63** μs	NA	NA	NA	NA	NA	NA	NA	NA	NA
[23]	×	×	1.0	NA	NA	NA	NA	NA	NA	-0.7 μs	0.64 μs	**1.52** μs	NA	NA	NA
[24]	×	×	1.0	109 ns	360 ns	**0.62** μs	316 ns	1.26 μs	**2.13** μs	NA	NA	NA	NA	NA	NA
non-PTP based clock synchronization solutions															
[26]	×	×	1.0	37.5 ms	6.80 μs	**37.6** ms	NA	NA	NA	NA	NA	NA	NA	NA	NA
[29]	×	×	1.0	6.00 μs	341 ns	**6.44** μs	21.0 μs	1.14 μs	**22.5** μs	NA	NA	NA	NA	NA	NA

output of both AP and clients. The added amplifier boosts the transmitter power by 18 dB, which gives sufficient power to form connection between the clients and AP.

The duration of the measurements is set to 20 minutes with the CS error stamped once per second. Similar to the single-hop network experiment, the CS performance is also quantified with both no network load and network load applied. For the network load setting, first two TCP iperf streams are simultaneously enabled from both Wi-Fi clients to the AP. Then the TCP streams are replaced by UDP streams, with a target throughput of 7 Mb/s on both clients. Note that 7 Mb/s is the average throughput observed in the TCP measurement.

The μ, σ and P_{90} values between each hop within the multi-hop hybrid network is shown in Table. II and Fig. 10. Like the conclusions from the single-hop network, it can be observed that the impact of network load on CS error between each client and AP is almost negligible. However, the CS error between each client and AP is relatively larger compared to the single-hop network case with no network load applied. In another word, we observed that the CS performance of the wireless network degrades when the AP is configured as a BC compared to the situation when the AP is acting as the GM. Intuitively, the AP clock is varied periodically when it is acting as a BC, making it more challenging for the clients to synchronize, hence increasing the CS error of the downstream network. However, even with the traffic load and AP configured as a BC, the single hop CS error between AP and client is still well within the 5.5 μs limit required by the *Wi-Fi Timesync* certification.

D. Performance comparison of the proposed PTP with the existing PTP solutions

A detailed CS performance comparison of the proposed PTP design with the existing PTP and non-PTP based CS solutions over WLAN is depicted in Table. III. The NA (i.e., Not Available) in Table. III corresponds to a metric which is not available for a solution in the literature. The column *Sync interval* indicates the time interval in seconds after which the synchronization procedure (i.e., *Sync,FollowUp* message exchange in case of PTP solution and *Beacon* broadcast in case of non-PTP solutions) is repeated. For a fair comparison of our validation, we have also estimated the P_{90} (see bold values in Table. III) of the existing solutions using Eq. 7.

Generally, most solutions have not evaluated the impact of traffic load, and neither the performance in a more complex network topology. The exceptions are [23], [24], [29]. [24] and [29] are the only solutions which provide the information pertaining to the CS performance in the presence of network load. Unlike our solution, it is shown that the CS performance of these solutions is significantly degraded in the presence of network load. That is, The CS error's σ increases from 360 ns to 1.26 μs in [24] and from 341 ns to 1.14 μs in [29]. [23] is the only solution other than ours that has evaluated CS performance in a network topology involving multi-hop with wired network segment. The CS performance of this solution is, apparently, better than ours in the absence of network load. The work has however used SW TS in the wireless part of the network and the performance of their work is not tested in the presence of non-PTP background traffic. Though the authors do mention that the solution is susceptible to non-PTP background traffic due to SW TS. Further, due to SW TS, the CS performance of all the above mentioned solutions is likely to suffer from propagation delay asymmetry generated from different frame sizes, modulation and coding scheme [13].

Purely looking at performance in a single hop network, the HW TS based PTP solutions significantly outperform our solution. This is mainly due to the TSF clock configured as PTP HW clock in our validation has TS resolution of one μs. Contrarily, the HW TS based solutions have utilized a dedicated HW PTP clock having better clock resolution in the range of ns. A secondary factor is the sync interval, [17] performs synchronization every 100 ms instead of 1 second, which improves the CS accuracy at the cost of traffic overhead. Thus, improved clock resolution of TSF clock, or reduced sync interval can enhance the CS performance of our design, which are a part of our consideration for future exploration.

V. CONCLUSIONS

In this paper, a new approach for PTP with HW TS is designed and verified over *openwifi*, an open-source IEEE 802.11 implementation on SDR. The *linuxptp* application and PHC subsystem of Linux kernel are employed to realize the

PTP software stack. Instead of adding a dedicated hardware clock, the existing TSF hardware of *openwifi* is used as the PTP HW clock. During the clock synchronization process, the default implementation of TSF however can only perform clock offset correction, leaving the clock skew uncorrected. We have modified the TSF hardware, enabling it to conduct both clock offset and skew correction. Thus, our approach allows the Wi-Fi chip manufacturers to enable PTP functionalities in their products with minimal hardware changes. It is shown that the approach is able to achieve mean synchronization accuracy of 0.018 μs, with standard deviation of 0.58 μs. The 90th percentile of the absolute CS error of our work is 1.25 μs, well below the 5.5 μs requirement of *Wi-Fi TimeSync* certificate introduced by the Wi-Fi alliance. In addition, the impact of traffic load in WLAN on the clock synchronization accuracy is also investigated and proven to be insignificant. Lastly, the performance of our approach is also tested on a multi-hop hybrid network with more Wi-Fi stations and wired network segment involved.

It is observed that the performance of our approach relatively degrades in multi-hop network. A stable TSF clock with high resolution can potentially enhance the performance of our approach. To this end, we consider to improve the performance by tuning the hardware clock at finer granularity (i.e., sub-microsecond level) without compromising the hardware utilization in the future. Though our novel approach for clock synchronization is validated on IEEE 802.11 standard, it is not specific for this standard. In other words, the methodology to support skew correction in an existing timer with support for PTP software stack can be applied on any wired or wireless standard incorporating embedded timer (e.g., 28 bits internal timer in Bluetooth).

ACKNOWLEDGMENT

This research was funded by the Flemish FWO SBO S003921N VERI-END.com (Verifiable and elastic end-to-end communication infrastructures for private professional environments) project and the Flemish Government under the "Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen" program.

REFERENCES

[1] O. Seijo, I. Val, J. A. Lopez-Fernandez, and M. Velez, "Ieee 1588 clock synchronization performance over time-varying wireless channels," in *2018 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)*. IEEE, 2018, pp. 1–6.

[2] "Ieee standard for local and metropolitan area networks—timing and synchronization for time-sensitive applications in bridged local area networks, ieee 802.1as," IEEE, 2007.

[3] B. Chen, Y.-P. Chen, J.-M. Xie, Z.-D. Zhou, and J.-M. Sa, "Control methodologies in networked motion control systems," in *2005 International Conference on Machine Learning and Cybernetics*, vol. 2. IEEE, 2005, pp. 1088–1093.

[4] "1588-2008 - ieee standard for a precision clock synchronization protocol for networked measurement and control systems," IEEE, 2008.

[5] R. Holler, T. Sauter, and N. Kero, "Embedded synutc and ieee 1588 clock synchronization for industrial ethernet," in *EFTA 2003. 2003 IEEE Conference on Emerging Technologies and Factory Automation. Proceedings (Cat. No. 03TH8696)*, vol. 1. IEEE, 2003, pp. 422–426.

[6] G. Gaderer, R. Holler, T. Sauter, and H. Muhr, "Extending ieee 1588 to fault tolerant clock synchronization," in *IEEE International Workshop on Factory Communication Systems, 2004. Proceedings.* IEEE, 2004, pp. 353–357.

[7] Cip sync, an ethernet based commercial product compliant with ieee 1588 standard. [Online]. Available: https://www.odva.org/technology-standards/distinct-cip-services/cip-sync/

[8] J. Feld, "Profinet-scalable factory communication for all applications," in *IEEE International Workshop on Factory Communication Systems, 2004. Proceedings.* IEEE, 2004, pp. 33–38.

[9] G. Venkatesan, "Avnu alliance ® white paper wireless tsn – definitions , use cases & standards roadmap," 2020.

[10] "Ieee standard for local and metropolitan area networks–timing and synchronization for time-sensitive applications," *IEEE Std 802.1AS-2020 (Revision of IEEE Std 802.1AS-2011)*, pp. 1–421, 2020.

[11] W. Chen, J. Sun, L. Zhang, X. Liu, and L. Hong, "An implementation of ieee 1588 protocol for ieee 802.11 wlan," *Wireless networks*, vol. 21, no. 6, pp. 2069–2085, 2015.

[12] A. Mahmood, G. Gaderer, H. Trsek, S. Schwalowsky, and N. Kerö, "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*. IEEE, 2011, pp. 13–18.

[13] 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*, vol. 13, no. 2, pp. 907–922, 2016.

[14] A. Mahmood, R. Exel, and T. Sauter, "Impact of hard-and software timestamping on clock synchronization performance over ieee 802.11," in *2014 10th IEEE Workshop on Factory Communication Systems (WFCS 2014)*. IEEE, 2014, pp. 1–8.

[15] 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*, vol. 56, no. 5, pp. 1632–1639, 2007.

[16] 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 & Metropolitan Area Networks*. IEEE, 2005, pp. 6–pp.

[17] 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*. IEEE, 2012, pp. 1–6.

[18] Intel. wi-fi 6 ax200 module. [Online]. Available: https://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/wi-fi-6-ax200-module-brief.pdf

[19] Intel. wi-fi 6 ax201 module. [Online]. Available: https://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/wi-fi-6-ax201-module-brief.pdf

[20] J. Xianjun, L. Wei, and M. Michael. (2019) open-source ieee802.11/wi-fi baseband chip/fpga design. [Online]. Available: https://github.com/open-sdr/openwifi

[21] R. Cochran and C. Marinescu, "Design and implementation of a ptp clock infrastructure for the linux kernel," in *2010 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*. IEEE, 2010, pp. 116–121.

[22] "Wi-fi certified timesync technology overview," Wi-Fi Alliance, 2017.

[23] A. Mahmood and F. Ring, "Clock synchronization for ieee 802.11 based wired-wireless hybrid networks using ptp," in *2012 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication Proceedings*, 2012, pp. 1–6.

[24] 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*, vol. 10, no. 2, pp. 1198–1206, 2014.

[25] D. Cavalcanti, J. Perez-Ramirez, M. M. Rashid, J. Fang, M. Galeev, and K. B. Stanton, "Extending accurate time distribution and timeliness capabilities over the air to enable future wireless industrial automation systems," *Proceedings of the IEEE*, vol. 107, no. 6, pp. 1132–1152, 2019.

[26] A. Mahmood, G. Gaderer, and P. Loschmidt, "Software support for clock synchronization over ieee 802.11 wireless lan with open source drivers," in *2010 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*. IEEE, 2010, pp. 61–66.

[27] A. Mahmood, R. Exel, and T. Bigler, "On clock synchronization over wireless lan using timing advertisement mechanism and tsf timers," in

2014 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS). IEEE, 2014, pp. 42–46.
[28] H. Puttnies, P. Danielis, A. R. Sharif, and D. Timmermann, "Estimators for time synchronization—survey, analysis, and outlook," *IoT*, vol. 1, no. 2, pp. 398–435, 2020.
[29] A. Mahmood, R. Exel, and T. Sauter, "Performance of ieee 802.11's timing advertisement against synctsf for wireless clock synchronization," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 1, pp. 370–379, 2016.
[30] "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 802.1as," IEEE, 2016.
[31] R. Cochran *et al.* (2015) The linux ptp project. [Online]. Available: http://linuxptp.sourceforge.net
[32] Xilinx. amba axi4 interface protocol. [Online]. Available: https://www.xilinx.com/products/intellectual-property/axi.html
[33] Xilinx. zc706, an evaluation kit for zynq-7000 soc. [Online]. Available: https://www.xilinx.com/products/boards-and-kits/ek-z7-zc706-g.html
[34] Analog devices. user guide of ad-fmcomms2-ebz an fmc board for the ad9361. [Online]. Available: https://wiki.analog.com/resources/eval/user-guides/adfmcomms2-ebz
[35] Xilinx. zedboard, a development board for zynq-7000 soc. [Online]. Available: https://www.xilinx.com/products/boards-and-kits/1-8dyf-11.html
[36] Xilinx. an overview of zynq-7000 soc data sheet. [Online]. Available: https://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview
[37] X. Jiao, W. Liu, M. Mehari, M. Aslam, and I. Moerman, "openwifi: a free and open-source ieee802. 11 sdr implementation on soc," in *2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring)*. IEEE, 2020, pp. 1–2.
[38] S. Bouckaert, W. Vandenberghe, B. Jooris, I. Moerman, and P. Demeester, "The w-ilab. t testbed," in *International Conference on Testbeds and Research Infrastructures*. Springer, 2010, pp. 145–154.
[39] Time-sensitive networking solution for industrial iot. [Online]. Available: https://www.nxp.com/design/designs/time-sensitive-networking-solution-for-industrial-iot:LS1021A-TSN-RD
[40] Xilinx.zcu102, an evaluation kit of zynq® ultrascale+™ mpsoc. [Online]. Available: https://www.xilinx.com/products/boards-and-kits/ek-u1-zcu102-g.html

Muhammad Aslam was born in Bahawalpur, Pakistan, in 1990. He received the B.Sc. degree in electronics engineering from International Islamic University, Islamabad, Pakistan, in 2014, and the M.S. degree in electronics and telecommunication engineering from the University of Kocaeli, Turkey, in 2017. He is currently pursuing the Ph.D. degree with the IDLab, a core research group of IMEC with research activities embedded in Ghent University, Belgium. His current research interests include wireless communication standards, video coding/motion estimation, and their real-time implementation on SDR platforms.

Wei Liu Wei Liu was born in China in 1986. She received the master's degree in electronic engineering from the University of Leuven, Campus GroepT, in 2010, and the Ph.D. degree from the IDLab, a core research group of IMEC with research activities embedded in Ghent University and the University of Antwerp, in 2016. During her doctoral education, she participated in multiple research projects, she became familiar with several software-defined radio platforms, and gained experiences in wireless testbed operations. She is a Post-Doctoral Researcher with Ghent University. Her research is conducted in the field of cognitive radio, focusing on spectrum analysis and interference prevention.

Xianjun Jiao Xianjun Jiao received his bachelor degree in Electrical Engineering from Nankai university in 2001 and Ph.D degree on communications and information system from Peking University in 2006. After his studies, he worked in industrial research institutes and product teams in the domain of wireless technology, such as Radio System Lab of Nokia Research Center(senior researcher), devices department of Microsoft (senior researcher) and Wireless Software Engineering department of Apple (RF software engineer). In 2016, he joined IDLab, a core research group of imec with research activities embedded in Ghent University and University of Antwerp. He is working as postdoc researcher at Ghent University on flexible real-time SDR platform. His main interests are SDR and parallel/heterogeneous computation in wireless communications. On his research track, 20+ international patents and papers have been authored/published.

Jetmir Haxhibeqiri received the Masters degree in Engineering (information technology and computer engineering) from RWTH Aachen University, Germany (2013). In 2019, he obtained a PhD in Engineering Computer Science from Ghent University with his research on flexible and scalable wireless communication solutions for industrial warehouses and logistics applications. Currently he is a postdoc researcher in the Internet Technology and Data Science Lab (IDLab) of Ghent University and imec. His current research interests include wireless communications technologies (IEEE 802.11, IEEE 802.15.4e, LoRa) and their application, IoT, wireless time sensitive networking, in-band network monitoring and wireless network management.

Gilson Miranda Jr. holds B.Sc. and M.Sc. degrees in Computer Science from Federal University of Lavras (UFLA), Brazil. In 2017 started his PhD with the Federal University of Minas Gerais (UFMG), Brazil, and is now pursuing a Joint PhD with University of Antwerp, Belgium, where he is carrying his research with IDLab. His main research interests are programmable networks, wireless networks, and machine learning applied to network management.

Jeroen Hoebeke received the Masters degree in Engineering Computer Science from Ghent University in 2002. In 2007, he obtained a PhD in Engineering Computer Science from Ghent University with his research on adaptive ad hoc routing and Virtual Private Ad Hoc Networks. Current, he is an associate professor in the Internet Technology and Data Science Lab of Ghent University and imec. He is conducting and coordinating research on wireless (IoT) connectivity, embedded communication stacks, deterministic wireless communication and wireless network management. He is author or co-author of more than 150 publications in international journals or conference proceedings.

Johann Marquez-Barja is a Professor at University of Antwerp, as well as a Professor in IMEC, Belgium. He is leading the Wireless Cluster at IDLab/imec Antwerp. He was and is involved in several European research projects with leading roles. He is a member of ACM, and a Senior member of the IEEE Communications Society, IEEE Vehicular Technology Society, and IEEE Education Society where he participates in the board of the Standards Committee. His main research interests are: 5G advanced architectures including edge computing; flexible and programmable future end-to-end networks; IoT communications and applications. He is also interested in vehicular communications, mobility, and smart cities deployments. Prof. Marquez-Barja is co-leading the Citylab Smart City testbed, part of the City of Things programme, and the SmartHighway testbed, both located in Antwerp, Belgium. He has given several keynotes and invited talks in different major events, as well as received 30 awards in his career so far, and co-authored more than 180 published works including publications, editorials, and books. He is also serving as Editor and Guest editor for different International Journals, as well as participating in several Technical Programme and Organizing Committees for several worldwide conferences/congresses.

Ingrid Moerman received her degree in Electrical Engineering (1987) and the Ph.D. degree (1992) from the Ghent University, where she became a part-time professor in 2000. She is a staff member at IDLab, a core research group of imec with research activities embedded in Ghent University and University of Antwerp. Ingrid Moerman is coordinating the research activities on mobile and wireless networking at Ghent University, where she is leading a research team of more than 30 members. Ingrid Moerman is also Program Manager of the 'Deterministic Networking' track at imec and in this role she coordinates research activities on end-to-end wired/wireless networking solutions driven by time-critical applications that have to meet strict QoS requirements in terms of throughput, latency bounds and dependability in smart application areas like Industry 4.0, vehicular networks and professional entertainment. Her main research interests include cooperative and intelligent radio networks, real-time software defined radio, time-sensitive networks, dynamic spectrum sharing, coexistence across heterogeneous wireless networks, vehicular networks, open-source prototyping platforms, software tools for programmable networks, next generation wireless networks (5G/6G/...), and experimentally supported research. Ingrid Moerman has a longstanding experience in running and coordinating national and EU research funded projects. At the international level, Ingrid Moerman was leading team SCATTER, consisting of researchers from IMEC-IDLab and Rutgers University (US), in the DARPA Spectrum Collaboration Challenge (SC2). The double prize winning SCATTER team was one of the 10 finalists at the DARPA SC2 championship event organized at Mobile World Congress in LA (October 2019).