Hardware Efficient Clock Synchronization across Wi-Fi and Ethernet Based 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 HW overhead. Moreover, the performance of the existing PTP solutions is mostly evaluated in single-hop wireless networks, while the performance across wired and wireless 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 solution utilizes the Timing Synchronization Function (TSF) clock, an existing clock in IEEE802.11 standard for synchronization between access point and WLAN stations. The performance of the proposed solution is first investigated within a single-hop WLAN and then across wired-wireless networks. Experimental results unveil that 90% of the absolute clock synchronization error falls within 1.4 μs.**

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

I. Introduction

CLOCK Synchronization (CS) is one of the prominent technologies for 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. For example, audio or videos streaming over distributed networks, and network based motion control [1]. Precision Time Protocol (PTP) is a state-of-the-art CS protocol introduced in the IEEE 1588 standard [2]. It is the de facto CS protocol in wired networks and is capable of providing sub-microsecond CS accuracy. The CS process in PTP is typically accomplished in two steps: (i) establishing master-slave hierarchy wherein the nodes in a network leverage the best master clock algorithm to elect a node with the best quality clock as a master clock and all the rest of the nodes' clocks are slaves; (ii) synchronizing of the master-slave clocks in which the master clock periodically exchanges special messages with slave clocks, which later extract the master clock information from these messages and synchronize to it by computing the time difference.

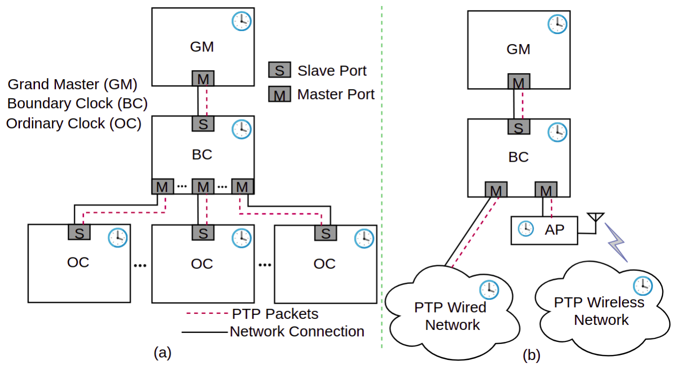

Fig. 1. An example of PTP based (a) conventional wired network and (b) wired-wireless hybrid network when configured in E2E mode.

The aforementioned process is not only applicable to 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 may consist of: (i) a Grand Master (GM) clock which is the primary reference clock for CS; (ii) one or more devices with multiple PTP ports fulfilled by either a Boundary Clock (BC) or a Transparent Clock (TC); and (iii) single-port Ordinary Clock (OC) which can only be used either as slave or master clock. The main difference between BC and TC is that in TC the PTP *Sync* packet is being forwarded to downstream PTP ports without touching the origin timestamp field, instead the correction field is being updated to incorporate propagation delay and residence time inside the device, the slave PTP port then uses the correction field to update the origin timestamp; whereas in BC the origin timestamp is being updated directly in the packet. Hence, BC and TC are mathematically equivalent. An example of a practical multi-hop PTP based CS network formed by BC is shown in Fig. 1-a.

In addition, depending on how the link delay between a PTP master and slave is measured, a multi-hop PTP network can be realized in End-to-End (E2E) or Peer-to-Peer (P2P) mode. In case of E2E mode, the slave generates requests to

Manuscript received April 6, 2021; revised July 22, 2021 and September 23, 2021; accepted September 30, 2021. Date of publication xxxx xx, 202x; date of current version October, 2021.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.

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).

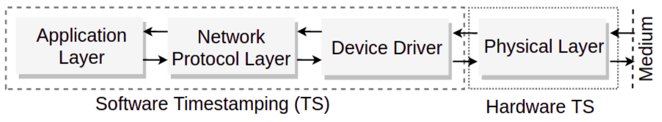

Fig. 2. Locations for timestamping in clock synchronization protocols.

measure link delay towards the master; whereas in P2P mode, a device (can be a slave or bridge) generates requests towards its directly connected device, and obtain the propagation delay of only one hop. The total link delay in P2P mode is the accumulated residence time and propagation time over all the devices between a PTP slave and master. Both modes have pros and cons, P2P mode offers more accurate link delay measurement hence better CS performance, however it requires all network devices to be PTP capable; whereas E2E mode can tolerate none PTP supporting devices, but the obtained CS performance is usually worse over larger network scale, especially when non-PTP supporting devices are present.

The CS accuracy of PTP is defined as the remaining time difference between a master and a slave which can not be further reduced. The CS accuracy of PTP depends on the timestamps captured at the moment of packet reception or transmission. Fig. 2 shows the possible locations for Timestamping (TS) in PTP. TS in PTP can be achieved via Hardware (HW) or Software (SW). HW TS is produced at or close to the physical layer, 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 network stack without HW assistance. The CS accuracy is significantly affected by the TS location, it is desired to place TS location as close as possible to the physical layer, so that the TS is least affected by the time variation of packets going through the network stack. Hence, 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 [3]–[5]. CIPSync is a PTP compliant industrial solution which provides CS accuracy of a few nanoseconds over conventional Ethernet [4]. Another example is PROFINET, it uses PTP as CS protocol over industrial Ethernet [5]. The industry however is increasingly interested in extending the PTP from wired network to wireless in the form of a hybrid multi-hop network for more flexibility, increased scalability and reduced deployment cost [6]. An example of a simple hybrid multi-hop PTP network is shown in Fig. 1-b.

Academic researchers have proposed several wireless CS solutions [7]–[13]. SW TS based PTP solutions are easy to realize, but they provide relatively poor CS accuracy [7]–[9]. Regarding the realization of HW TS based PTP network, 802.1AS [14], a specific profile of PTP referred to as the generalized PTP (gPTP) is commonly used in industrial applications. 802.1AS provides approaches to maintain high precision CS across Ethernet and Wi-Fi, leveraging on specific 802.11 frames in the Timing Measurement (TM) or Fine Timing Measurement (FTM) to trigger TS in Wi-Fi cards. So far only Intel claims to have working CS solution over Wi-Fi cards with this approach [15], [16], though it seems to be specific to Windows operating system and has very limited information exposed. Further exploration regarding this approach is detailed in Section II. [17] uses TM frames based 802.1AS for CS over Intel Wi-Fi 5 cards in Collaborative Robotics applications. Though this work focuses on combining robotic operating system with wireless TSN traffic flow to maintain optimal network jitter and latency, it never quantified CS performance directly, and neither exposed any information regarding how 802.1AS is realized over Wi-Fi TM. Due to the rather rare and limited FTM/TM support in available Wi-Fi cards, most academic solutions for HW TS have not followed this approach [10]–[13], nevertheless the results of these work demonstrate that HW TS based PTP gives better performance, though, at the cost of dedicated HW added to realize TS. Moreover, the CS performance of these PTP solutions is mostly analysed only in a single-hop wireless network. The support of PTP BC/TC is not mentioned, therefore the performance of these solutions over a combined wired and wireless network is either taken for granted or simply left as future work.

In this paper, we enable HW TS based PTP in *openwifi* [18], an open-source Wi-Fi chip design. Instead of using an additional clock for HW TS, we leverage the existing Timing Synchronization Function (TSF) clock with minimal modifications in Field Programmable Gate Array (FPGA) to achieve the same purpose. 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 [19]. Lastly, the performance of our solution is characterized over both single-hop and multi-hop network across wired and wireless network domains.

The rest of the paper is structured as follows. The state of art of standardization, available tools and related work is detailed in Section II. Section III describes the proposed 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

The purpose of this work is to provide a PTP based CS solution across Wi-Fi and Ethernet network. In this section, we (i) examine the relevant standardization, (ii) identify potential solutions based on the standards and available resources, (iii) present a comparison of the selected approach against existing work, and (iv) finally summarize the contribution of this paper.

A. Standardization

The Wi-Fi alliance has introduced the *Wi-Fi TimeSync* certificate [20], which recommends to follow the IEEE 802.1AS (or gPTP) for clock synchronization over Wi-Fi. Some of the key differences between the gPTP and the original PTP protocols are that gPTP excludes usage of BC, it uses a TC to operate as a bridge; and all devices involved in gPTP network should provide HW TS and operate in P2P mode.

Unlike the original PTP protocol, 802.1AS separates the medium independent and medium dependent layers. Medium independent layer is common for all network types, which uses timestamps provided by medium dependent layer to perform

CS; the medium dependent layer on the other hand is different for various network types, such as Ethernet and Wi-Fi. For Ethernet, the packets being timestamped are the event packets defined in the PTP protocol; whereas for Wi-Fi, the 802.11 Mac Layer Management Entity (MLME) is responsible for generating packets defined in TM to trigger TS in the Wi-Fi card. Later on the 802.11 standard introduces FTM, which is subsequently included in 802.1AS-rev.

FTM and TM both are expected to provide timestamps that indicate the start of preamble of an Wi-Fi packet accurately, which serve as a way for Wi-Fi to support HW TS required by 802.1AS. Since functionally FTM and TM are equivalent, we focus our discussion on TM and the original 802.1AS standard in subsequent sections.

B. Available resources and potential solutions

In this section, we identify potential solutions and their available and missing features, focusing on: (i) the available software to realize 802.1AS over Wi-Fi, (ii) the support of 802.11 TM and PHC in the OS, and (iii) the necessary features in Wi-Fi driver and hardware to support the operation. Regarding the software tools, we mainly explore whether the application supports 802.1AS wireless port (i.e., the interface to exchange timestamps with 802.11 TM) and the required 802.1AS roles (i.e. end stations and bridges). Regarding the support of PHC in OS, it comes down to two aspects, namely the capability to pass HW timestamp through the OS to the application, and the capability to control the hardware clock in a PTP compliant way.

gptp[1] is a gPTP daemon managed by the AVNU alliance, its development is mainly lead by Intel. The daemon is an open source implementation of PTP software stack of 802.1AS for Windows and Linux OS. Though upon further exploration, we find that the daemon does not support the 802.1AS wireless port in Linux OS. The authors in [21] attempt to run gptp daemon with TM as a hardware stack on Intel 8260 AC [22] in Windows OS. However, due to non-native support of PTP hardware clock in Windows OS, they were unable to measure the CS accuracy. The support of PTP hardware clock is well present in the Linux OS, its PHC subsystem allows control of PTP hardware clock, and specific socket options are provided to pass the timestamps through the OS. As TM packets are produced and consumed in the 802.11 MLME, we further explore the support of TM in mac80211 subsystem of Linux kernel, the conclusion is however that the kernel does not provide support for constructing TM packet. The interface between kernel and Wi-Fi driver only defines functions to trigger measurement and get results. In addition, the structure to return TM results does not contain timestamps, it only passes the averaged round trip time to the upper layers. This means without kernel modification, even if a Wi-Fi card supports TM, the result obtained can only be used for ranging application, not CS. In short, if we want to accomplish our goal using gptp daemon in Windows, we need to add hardware PHC support in Windows OS; if we proceed with gptp in Linux, we need to extend the gptp application to support wireless port and modify the Linux kernel for TM support. Both approaches already require a considerable amount of work in pure software stack, and above that the driver and hardware should be able to construct TM packet and timestamp it.

[1]https://github.com/Avnu/gptp

The *linuxptp* is another open source implementation of the PTP software stack in Linux OS. It is a widely used software stack in both academia and industry for CS in Ethernet based distributed systems. Though it supports both P2P mode and E2E modes, it only supports the IEEE 802.1AS in the role of end station. In other words, the default *linuxptp* cannot fulfill the role a 802.1AS bridge in a multi-hop network. In addition, *linuxptp* replies on standard PTP packets to obtain the timestamps, the software does not include possibility to leave 802.11 MLME to handle this task. Hence the support of 802.1AS wireless port needs to be added to the application. In short, if we want to accomplish our goal using *linuxptp* in a fully 802.1AS compliant way, we need to extend the application to support 802.1AS bridge and wireless port, modify the Linux kernel for passing timestamps from the 802.11 TM, and finally construct TM packet and timestamp it in driver and hardware of Wi-Fi interface.

Inspired by Ethernet based PTP solutions, one can also achieve accurate CS in multi-hop network with merely PTP packets. Although *linuxptp* does not support the 802.1AS bridge (i.e. a TC operating in 2-step P2P mode), it does support regular IEEE 1588 BC/TC. We then come up with configuring AP as a BC using *linuxptp* in the E2E mode, combined with TS the PTP event packets in Wi-Fi hardware and driver. BC in E2E mode is preferred for two reasons: (i) all PTP packets of different network interfaces are generated and consumed independently, no bridging is required, hence simpler in terms of network configuration; (ii) Wi-Fi AP and clients are almost always connected, using P2P mode will lead to complexity such as the support of residence time calculation but without much performance gain. As introduced previously, BC and TC are mathematically equivalent, and the difference mainly shows when non-PTP aware devices are involved, we believe the performance penalty of using E2E BC instead of P2P TC will be trivial in wireless network domain. The same conclusion applies for replacing TM packets by PTP event packets. A summary of the pros and cons of the potential solutions are given in Table I, where a "$Need$" indicates a feature is required and available, a "$Need$*" indicates a feature is required but missing, and "$NoNeed$" indicates a feature is not needed for an approach. We observe that the last approach only requires the Wi-Fi interface to TS regular PTP packets, whereas all the rest requires some level of modification in the application and OS, in addition to the TM packet construction and TS support. Although this approach is not fully 802.1AS compliant, we believe the gain of the reduced implementation effort and the convenience to achieve PTP across Ethernet and Wi-Fi with standard OS and application significantly outweigh the drawback. Hence this is the selected approach.

C. Comparison against existing work

There have been many attempts to implement PTP over IEEE 802.11 Wireless Local Area Network (WLAN). These

TABLE I
COMPARISON OF POTENTIAL WAYS TO REALIZE PTP ACROSS WIRED AND WIRELESS NETWORK, $Need$ INDICATES A FEATURE IS REQUIRED AND AVAILABLE, $Need^*$ INDICATES A FEATURE IS REQUIRED BUT MISSING, AND $NoNeed$ INDICATES A FEATURE IS NOT NEEDED FOR AN APPROACH.

	Application		Operating System		Driver & HW		
	802.1AS Wireless Port	802.1AS all roles	PHC support	TM support	TS PTP PKT	Construct TM PKT	TS TM PKT
gptp in Windows	$Need$	$Need$	$Need^*$	$Need^*$	$NoNeed$	$Need^*$	$Need^*$
gptp in Linux	$Need^*$	$Need$	$Need$	$Need^*$	$NoNeed$	$Need^*$	$Need^*$
linuxptp 802.1AS TS TM	$Need^*$	$Need^*$	$Need$	$Need^*$	$NoNeed$	$Need^*$	$Need^*$
linuxptp E2E TS PTP [this work]	$NoNeed$	$NoNeed$	$Need$	$NoNeed$	$Need$	$NoNeed$	$NoNeed$

work can be categorized into SW TS based PTP, and HW TS based PTP implementations.

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. In addition to these metrics, the Wi-Fi Alliance has introduced the *Wi-Fi TimeSync* certificate to specify the requirement of CS performance between multiple Wi-Fi devices [20]. The certification requires the 90th percentile of the absolute CS error (P_{90}) to be within 5.5 μs of the observed time (i.e., 120 sec). In this paper, (1) is used to quantify the P_{90}

$$P_{90} = \begin{cases} i, & \text{if } f(i) = 0.9 \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

$$f(i) = \frac{1}{N}\sum_{n=1}^{N} 1_{C_{err}(n)} \quad where$$
$$1_{C_{err}(n)} = \begin{cases} 1, & \text{if } |C_{err}(n)| \leq i \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

where, i value denotes the P_{90} when $f(i)$ (empirical cumulative distribution function) is equal to 0.9. To check the compliance with *Wi-Fi TimeSync* certificate, We include P_{90} together with μ and σ metrics to measure the CS performance of our solution in a network.

1) PTP with software timestamping: [7] proposes a PTP solution over IEEE 802.11 WLAN with SW TS at the application layer. To mitigate the impact of asymmetric bidirectional delay of WLAN on CS accuracy, they first designed a delay filtering algorithm based on Kalman filter, and then introduced a modified Proportional Integral (PI) controller based clock servo system. [7] is validated on a Linux based embedded development board. The μ is limited to -14.24 μs with a σ of 27.65 μs. In [8], PTP SW TS is done inside the interrupt service routine of the Ath5k Wi-Fi card[2] driver of Atheros AR5xxx chipset in Linux. However, the mean CS accuracy of this work is still 6.60 μs with σ of 0.58 μs [23]. Another design of PTP over IEEE 802.11b WLAN is realized in a Linux Personal Computer (PC) [12]. They have made changes in the radio driver to achieve SW TS, leading to better CS accuracy (i.e., μ: 4.6 μs, σ:1.58 μs) when configured in Access Point (AP) mode. An effort is made in [9] 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.

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

To summarize, all these SW TS based solutions have been realized over commercial off-the-shelf WLAN chipsets, they provide decent CS accuracy (in the order of several microsecond). However, the performance is evaluated in rather simple scenario (e.g., no traffic load and mostly over single-hop network). Additionally, propagation delay asymmetry caused by different frame sizes and Modulation and Coding Scheme (MCS) can potentially harm CS performance [23], [24]. Although these solutions provide important insights for PTP implementation in wireless network, we argue that they are inadequate for industrial applications.

2) PTP with hardware timestamping: [13] introduces a HW TS based PTP solution over IEEE 802.11b WLAN. The work has used a dedicated Adder Based Clock (ABC) in FPGA for HW TS. In the prototype, a customized version of the PTP protocol is used, instead of using standard PTP 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., μ is 0.24 ns with σ of 0.53 ns. Another HW TS based PTP solution is analyzed over WLAN by using an embedded processor and Programmable Logic Device (PLD) in [12]. To realize HW TS, they have implemented 4 hardware modules including two 64 bits counters on the PLD. The solution gives a μ of 1.1 ns with a σ of 1.76 ns, but it uses a custom Media Access Control (MAC) mechanism, which is not compatible with the IEEE 802.11 standard. Despite the good performance, the customization of PTP and Wi-Fi stack will hinder the applicability of these solutions when needed to form a PTP based CS network across Wi-Fi and Ethernet with traffic load.

D. Contribution of this paper

1) This solution supports HW TS, making it more accurate than SW TS based PTP in terms of CS accuracy.
2) This work uses the existing TSF clock of Wi-Fi standard as PTP HW clock, rather than adding a new HW clock, making it more economical in terms of HW footprint.
3) Existing TSF based solutions [25], [26] only apply clock offset correction during the CS process. PTP however requires both clock offset and skew correction [27]. The CS process between the slave clock (C_s) and the master clock (C_m) can be modeled by (3)

$$C_s(t) = S \times C_m(t) + b \tag{3}$$

where S and b represent the skew and offset difference of the slave clock with respect to its master [28]. Clock offset is the relative time difference between master

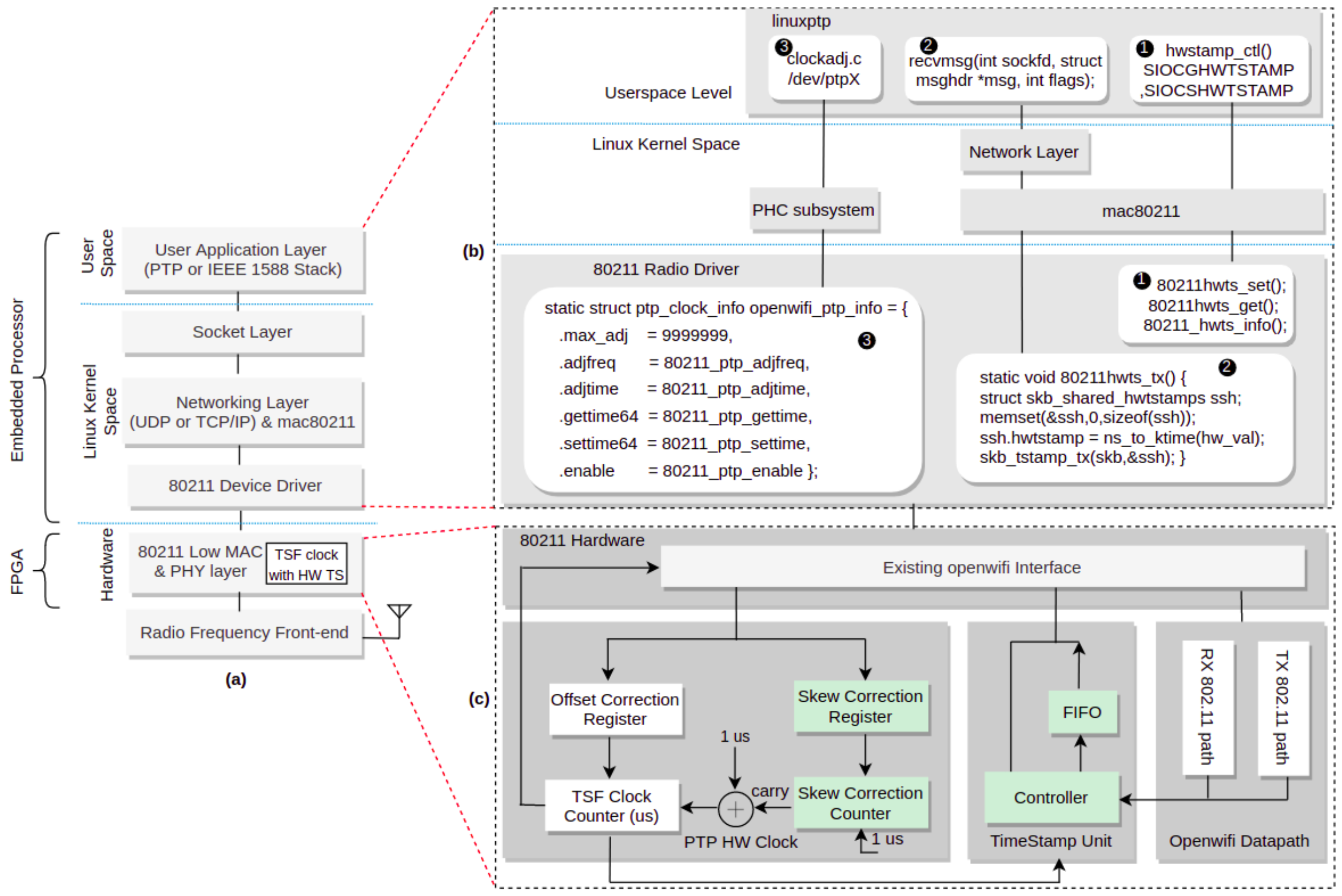
Embedded Processor
User Space
User Application Layer (PTP or IEEE 1588 Stack)
Socket Layer
Linux Kernel Space
Networking Layer (UDP or TCP/IP) & mac80211
80211 Device Driver
FPGA
Hardware
80211 Low MAC & PHY layer
TSF clock with HW TS
Radio Frequency Front-end
(a)
(b)
linuxptp
Userspace Level
clockadj.c /dev/ptpX
recvmsg(int sockfd, struct msghdr *msg, int flags);
hwstamp_ctl() SIOCGHWTSTAMP ,SIOCSHWTSTAMP
Linux Kernel Space
Network Layer
PHC subsystem
mac80211
80211 Radio Driver
static struct ptp_clock_info openwifi_ptp_info = { .max_adj = 9999999, .adjfreq = 80211_ptp_adjfreq, .adjtime = 80211_ptp_adjtime, .gettime64 = 80211_ptp_gettime, .settime64 = 80211_ptp_settime, .enable = 80211_ptp_enable };
80211hwts_set(); 80211hwts_get(); 80211_hwts_info();
static void 80211hwts_tx() { struct skb_shared_hwtstamps ssh; memset(&ssh,0,sizeof(ssh)); ssh.hwtstamp = ns_to_ktime(hw_val); skb_tstamp_tx(skb,&ssh); }
(c)
80211 Hardware
Existing openwifi Interface
Offset Correction Register
Skew Correction Register
1 us
carry
TSF Clock Counter (us)
Skew Correction Counter
1 us
PTP HW Clock
FIFO
Controller
TimeStamp Unit
RX 802.11 path
TX 802.11 path
Openwifi Datapath

Fig. 3. Generalized architecture (a) of our proposed PTP design over WLAN, which is primarily composed of (b) software stack and (c) hardware unit.

and slave, whereas clock skew is the relative difference in clock frequency between the master and slave's clocks. As the clock skew between master and slave remains uncorrected in the existing TSF based solutions, the clocks quickly diverge after each correction. IEEE 802.11 [29] has specified ± 20 ppm frequency skew for WLAN chipsets, resulting CS errors of up to ± 40 μs per second if only offset correction is applied every second. Thus, we introduce skew correction feature combined with clock offset correction in a TSF clock by modifying the existing TSF block of *openwifi* in FPGA.

4) The introduction of skew correction in TSF may help to further enhance the performance of the existing applications relying on synchronized TSF clocks. Similarly, the 802.11 MAC sublayer Managment Entity (MLME) operations relying on the TSF synchronization could gain benefit from the more accurate TSF synchronization between client and AP in the same Basic Service Set (BSS). The enhanced TSF based synchronization can also be used to maintain the better coordination among APs in overlapping BSSs.
5) Our solution based on an open source radio chip design is full stack and Linux compatible, using existing PTP software and Linux kernel support.
6) Unlike existing work, we have quantified the CS performance in terms of P_{90} to validate the compliance of our solution with the *Wi-Fi TimeSync* certificate.
7) Lastly, to examine the adequacy of our solution for industrial applications, the CS performance is investigated over a single-hop wireless network as well as a network across wired and wireless network domains, coexisting with a large amount of non-PTP traffic.

III. The Proposed Solution

Fig. 3-a displays 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* is running. 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*[3] software as a userspace application. *Linuxptp* software is the Linux based PTP implementation for wired network. It relies on certain system calls to determine a device's TS capability (i.e., SW or HW TS), which are not accessible for Wi-Fi card driver, hence changes are made to bypass the check without hurting *Linuxptp*'s compliance with the PTP standard. We have further modified the radio device driver, making it able to interface with the PTP HW clock. Lastly, necessary changes are made in the existing TSF hardware to enable skew corrections, so

[3]http://linuxptp.sourceforge.net

TABLE II
HARDWARE UTILIZATION COMPARISON WHEN EITHER TSF OR A DEDICATED CLOCK IS USED AS PTP HW CLOCK.

Resources	LUTs	FFs
Default TSF	76	74
TSF as PTP HW Clock	177	130
Dedicated PTP HW Clock	154	136
HW Efficiency	23%	38%

that it can be used as a PTP HW clock. In short, the proposed PTP solution can be generalised into two steps: (1) design a wireless PTP software stack, and (2) modification of the existing TSF clock in HW to allow PTP HW TS. These two steps are detailed in this section.

A. Design of the PTP software stack

A diagram containing the main components used in the PTP software stack is shown in Fig. 3-b. There is an existing PTP Hardware Clock (PHC) subsystem in the Linux kernel. The PHC subsystem offers Application Programming Interfaces (APIs) for both userspace applications and device drivers to control the HW clock. Introducing PTP clock support for a WLAN interface requires the integration of these APIs together with TS at both userspace level and device driver.

We adopt *linuxptp* as the software stack of PTP in userspace. *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()* in userspace 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 and *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 ioctl system call to configure which outgoing and incoming packets should be timestamped. The packet selection is needed, as non-PTP packets on the network stacks requires no HW TS. The corresponding callback function implemented in our driver is *80211hwts_set* (see ① in Fig. 3-b). *SIOCGHWTSTAMP* is a system call 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*[4] *-T sdr0* command is executed, where *sdr0* represents the *openwifi* interface recognized by the Linux OS.
2) After configuring the HW timestamp settings for PTP packet, the next step is to enable the HW TS in the radio driver. *sk_buff* is a data structure in Linux networking stack 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. 3-b (see ② in radio driver).
3) The last step is to expose the HW clock to *linuxptp*, so that it can regulate the clock from userspace. 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. 3-b (see ③ in the radio driver).
The *80211_ptp_adjtime* and *80211_ptp_adjfreq* are the most important callback functions. The former function performs clock offset correction and the latter does skew correction of PTP HW clock. The clock skew correction value is calculated using (4).

$$S_c = \left(\frac{10^9}{F_r \times ppb}\right) \times 10^6 \tag{4}$$

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 microsecond 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 using (4) is 12,805 μs, which is later used by PTP HW clock (see Fig.3-c) to correct clock skew. Lastly, the ptp_clock is exposed as a character device ($/dev/ptpX$) to userspace, where *linuxptp* directly uses it to regulate the HW clock.

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

B. Design the assisting HW for PTP HW TS

As shown in Fig. 3-c, the assisting HW 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. The PTP software stack uses the existing *openwifi* interface to communicate with 802.11 datapath.

Instead of using a conventional way to design a PTP HW clock, our design realizes the PTP HW clock by leveraging the existing TSF clock. The motivation of this design choice is twofold: (i) modifying the existing TSF clock costs less hardware resources than adding an additional clock; (ii) applications relying on TSF for synchronization may also benefit from this approach. The Wi-Fi standard provides basic synchronization across the TSF clocks in a Basic Service Set (BSS), though there is no rate correction and neither link delay measurement, hence the CS accuracy of the Wi-Fi standard is worse than PTP. For instance, IEEE 802.11 has specified ± 20 ppm frequency skew for WLAN chipsets, which can additionally introduce CS errors of up to $\pm 40\ \mu s$ per second if basic TSF based CS (without rate correction) is applied every second. Despite of this fact, many researchers and developers take advantage of this feature. For instance, [30] uses the synchronized TSF in a Wi-Fi BSS for Time Division Multiple Access (TDMA), though due to the poor CS accuracy, the solution requires 20 μs guard interval in a TDMA slot. In [31], the synchronized TSF is used to coordinate transmission from a Wi-Fi BSS to avoid interfering with a sensor network, it is required to mute the Wi-Fi transmission somewhat in advance to account for the CS inaccuracy. On top of that, IEEE 802.11 standard also uses TSF for maintaining coordination between APs in overlapping BSS. With our approach, applications relying on TSF synchronization can greatly improve their performance without any additional effort. Hence we believe that the improved CS accuracy of TSF clocks in a Wi-Fi network is substantial.

While HW CS demands both clock offset and skew correction, default TSF design 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. 3-c). Table. II shows a comparison of hardware utilization when the default TSF clock (i.e., without skew correction capability), the modified TSF clock supporting skew correction feature, or a dedicated HW is allocated for the PTP HW clock. The HW efficiency in the Table. II is calculated using (5)

$$Efficiency = \left(\frac{HW_{ptp} + HW_{dftTSF} - HW_{TSF}}{HW_{ptp} + HW_{dftTSF}} \right) \times 100 \quad (5)$$

where HW_{dftTSF}, HW_{tsf} and HW_{ptp} represent the hardware consumed by the default TSF in *openwifi*, the modified TSF in our work, and a hardware clock allocated as dedicated PHC, respectively. The consumed hardware can be either Look Up Table (LUT) or Flip Flop (FF). Table. II shows that the modified TSF clock saves 23% LUTs and 38% FFs as compared to a dedicated HW PTP clock added on top of the default TSF.

The radio driver calculates the upper limit (i.e. the S_c value) for Skew Correction Counter (SCC) using (4), and loads the calculated value into the Skew Correction Register (SCR). 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 local oscillator. Upon skew correction, the TSF counter is either incremented by two or zero depending upon the sign of 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 ② in the radio driver in Fig. 3-b). The radio driver also performs a similar action for reception process, for simplicity, it is not shown in Fig. 3-b. The radio driver performs all these actions inside the Interrupt Service Routine (ISR). Subsequently, the *linuxptp* calls *recvmsg()* function to read these TS values and perform CS.

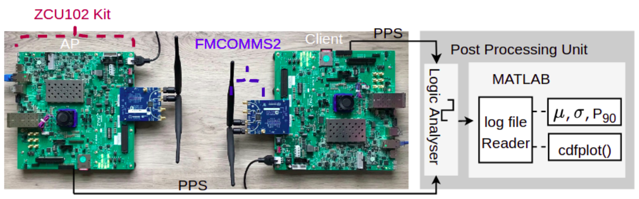

Fig. 4. Experimental setup of the single-hop network used to measure the performance of our proposed PTP over WLAN.

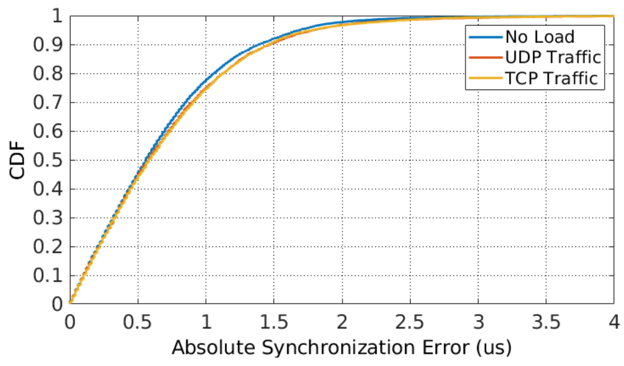

Fig. 5. Cumulative Distribution Function (CDF) of the absolute clock synchronization error of our proposed PTP solution over single-hop WLAN when no traffic load, UDP traffic and TCP traffic is applied.

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 Wi-Fi network and across Ethernet and Wi-Fi network. The CS performance of our design is quantified in terms of μ, σ and P_{90}. Lastly, the CS performance of our design is compared against the performance of existing CS solutions. Since the solutions in literature only provides information regarding μ and σ values, we have estimated P_{90} value with the help of (6). We assume that the samples in a CS error measurement follows Gaussian distribution, then its absolute value will follow folded Gaussian distribution, whose cumulative distribution function is given in Eq 6

$$f(C_{err}) = \frac{1}{2}\left[erf\left(\frac{C_{err}+\mu}{\sigma\sqrt{2}}\right) + erf\left(\frac{C_{err}-\mu}{\sigma\sqrt{2}}\right)\right] \quad (6)$$

$$erf\left(\frac{C_{err} \pm \mu}{\sigma\sqrt{2}}\right) = \frac{2}{\sqrt{\pi}} \int_0^{\left(\frac{C_{err}\pm\mu}{\sigma\sqrt{2}}\right)} e^{-t^2}\, dt \quad (7)$$

where, $f(C_{err})$ reflects the estimated percentile of absolute CS error at $|C_{err}|$th point and C_{err} represents CS error between the master and the slave clock. In other words, the $|C_{err}|$ value corresponds 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

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

Parameters	No load	UDP load	TCP load
μ	-0.279 μs	-0.330 μs	-0.325 μs
σ	0.820 μs	0.872 μs	0.868 μs
P_{90}	1.40 μs	1.48 μs	1.46 μs

between two *openwifi* Software Defined Radio (SDR) boards has been established, as illustrated in Fig. 4. The WLAN is set-up in infrastructure mode, where one SDR acts as the Access Point (AP) and the other behaves as a client. Due to the lack of Power Amplifier (PA) 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 in the particular experiment is composed of Zynq UltraScale+MPSoC ZCU102 Evaluation Kit[5], and FMCOMMS2[6], an analog RF frontend. The Zynq SoC on ZCU102 Evalutaion Kit further comprises Programmable Logic (PL) (FPGA) and Processing Subsystem (PS) (ARM Cortex-A53). 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* and PTP software stack are running on embedded PS part. The OS 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 MCS values for IEEE 802.11a is dynamically adapted up to 7 by Linux *mac80211* minstrel rate control algorithm.

The AP acts as the PTP master and the client is the PTP slave. In our setup, PTP is configured in E2E mode. The PTP messages in the *linuxptp* are transmitted or received using UDP/IPv4 via sockets API (see Fig. 3-a). The *linuxptp* v2.0 is used in the experiment, it is configured to perform synchronisation once per second. The CS accuracy is measured by analysing the Pulse Per Second (PPS) signal based on TSF clock from both slave and master with Saleae logic analyzer[7]. The sampling rate of the logic analyzer is configured at 100MHz, resulting in 10 ns resolution. Subsequently, the PPS signals captured with the help of the logic analyser are fed to post processing unit indicated in Fig. 4, where the CS performance metrics (i.e. μ, σ and P_{90}) are derived based on the difference between the PPS pulses.

B. Experimental Evaluation over single-hop WLAN

Before evaluating the CS accuracy over the single-hop WLAN, We first quantify the Convergence Time (CT) of our proposed PTP solution using experimental setup shown in Fig. 4. We define CT as a time when CS error or frequency offset(i.e., the measured clock skew between the PTP master and slave clocks in ppb) reaches a stable range which can be positive or negative depending on whether the slave clock is leading or lagging behind the master clock. Short CT is always

[5]https://www.xilinx.com/products/boards-and-kits/ek-u1-zcu102-g.html
[6]https://www.analog.com/en/design-center/evaluation-hardware-and-software/evaluation-boards-kits/eval-ad-fmcomms2.html
[7]Saleae logic analyzer https://www.saleae.com/

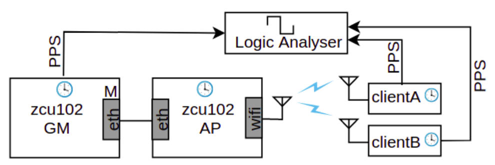

Fig. 6. Experimental setup used to measure the performance of our proposed PTP across wired-wireless network.

desired, since time critical applications can run only when CS error is stable and small enough. Experimental results show that CT takes ≈4-6 master-slave interactions on the typical SDR platform with oscillator frequency error of ±61.5 ppm[8] to get stabilized.

A realistic wireless network in the industrial environment contains background traffic in addition to PTP messages which can degrade the CS performance, due to the delay or dropping of PTP packets when encounter network congestion. To this end, the 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 in a more practical condition. The measurement is conducted in an office environment, the presence of other Wi-Fi devices does not show visible impact to the experiment result. The HW setup displayed in Fig. 4 is leveraged to quantify the CS error. Note that the experimental evaluation uses results of the stable phase of PTP. The measurements last for 20 minutes with the synchronization performed once per second between master and slave devices.

The CS errors quantified under different network loads are shown in Fig. 5 and Table. III. As seen from Fig. 5, the P_{90} is 1.40 μs with σ of 0.82 μs without traffic load. The iperf[9] software is used to generate traffic between the SDRs. First a TCP stream is enabled from the client to the AP accompanied by PTP message exchange. Since TCP is by design bi-directional, using TCP of iperf will automatically find the maximum throughput (i.e. 12 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 stable. The P_{90} rises to 1.46 μs with σ of 0.87 μs while running the TCP traffic (see Fig. 5). Then UDP traffic is enabled in similar way, UDP is unidirectional, the average throughput found in TCP is applied as the target throughput in UDP stream. The P_{90} jumps to 1.48 μs with σ of 0.87 μs when 12 Mb/s UDP iperf is enabled. The results show that the impact of network load on the CS error is almost negligible; i.e., the P_{90} only increases by 0.08 μs and 0.06 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 unveil that our solution is capable of providing a P_{90} much better than 5.5 μs, which is required for *Wi-Fi TimeSync* [20] certification, in both optimum and more practical scenarios.

[8]https://www.xilinx.com/products/boards-and-kits/ek-u1-zcu102-g.html
[9]Iperf command: https://iperf.fr/

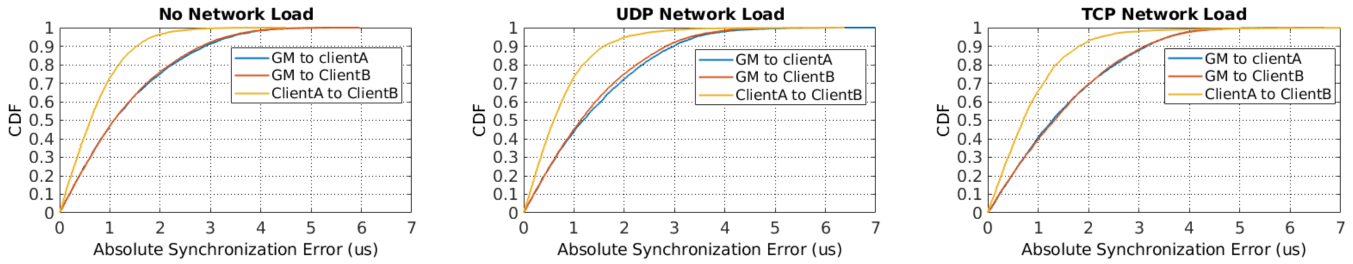

Fig. 7. Cumulative Distribution Function (CDF) of absolute clock synchronization error of our proposed PTP across wired and wireless network when no traffic load, UDP traffic and TCP traffic is applied.

TABLE IV
THE MEASURED CLOCK SYNCHRONIZATION PERFORMANCE OF OUR PROPOSED PTP ACROSS WIRED AND WIRELESS NETWORK.

Description	No load			UDP load			TCP load		
	μ (μs)	σ (μs)	P_{90} (μs)	μ (μs)	σ (μs)	P_{90} (μs)	μ (μs)	σ (μs)	P_{90} (μs)
ClientA to GM	0.94	1.42	2.88	0.98	1.49	2.98	0.90	1.65	3.16
ClientB to GM	0.87	1.43	2.81	0.93	1.42	2.82	0.87	1.68	3.10
ClientA to ClientB	-0.08	0.94	1.54	-0.05	1.02	1.66	-0.03	1.19	1.80

C. Experimental evaluation across WLAN and Ethernet

The experimental setup used to measure the CS performance across the Wi-Fi and Ethernet network is shown in Fig. 6. An additional Zynq UltraScale+ MPSoC ZCU102 Evaluation Kit (refferred to as ZCU102 hereafter) is used as PTP GM, as the board supports PTP HW TS and PPS on its Ethernet port. The *openwifi* AP composed of a ZCU102 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 GM and the HW clock attached to the *openwifi* interface functions as the master clock to the Wi-Fi clients. 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. Thanks to the selection of E2E mode, the PTP packets are sent in layer 3 encapsulation (i.e., IPv4/UDP packets). There is no bridging of PTP packets between Wi-Fi and Ethernet, each interface is an independent PTP port that generates and consumes PTP packets in its own IP subnet, which greatly simplifies the network configuration. The wireless network includes two Wi-Fi clients. Each client consists of a ZCU102 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 duration of the measurements is set to 20 minutes with the synchronization performed once per second between PTP master and slave devices. Similar to the single-hop WLAN, the CS performance is quantified both with and without network load. 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 8.5 Mb/s on each client (i.e., 8.5 Mb/s is the average throughput observed in the TCP measurement).

The μ, σ and P_{90} values between the GM and Wi-Fi client across Wi-Fi and Ethernet network is shown in Table. IV and Fig. 7. Comparing to the results obtained in single-hop WLAN, the P_{90} between GM and client has increased from 1.4 μs to 2.8 μs approximately. This change is mainly caused by the additional CS error being accumulated over the extra hop (i.e., AP acting as a BC in between GM and Wi-Fi client). Intuitively, the AP clock is varied periodically as a BC, making it more challenging for the clients to synchronize, hence increasing the CS error of the downstream network. Though, similar to the conclusions from the single-hop experiment, it can be observed that the impact of network load on CS error between each client and GM is almost negligible. In addition, we also measure the difference of PPS pulses between the two Wi-Fi clients, which is much smaller than the error between a client and GM. We believe this is important for applications such as a wireless speaker or synchronized movement of robotic arms. In this type of applications, the end goal is that the actions taken on multiple PTP slaves' are synchronized. In general, even with traffic load and AP configured as a BC in between the GM and Wi-Fi clients, the P_{90} of E2E CS error is still well within the 5.5 μs required by the *Wi-Fi Timesync* certificate for performance over a single hop WLAN.

D. Comparison against the existing PTP solutions

A detailed CS performance comparison of the proposed PTP design with the existing CS solutions over WLAN is depicted in Table. V. The NA (i.e., Not Available) in Table. V 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 is repeated. For a fair comparison of our validation, we have also estimated the P_{90} (see bold values in Table. V) of the existing solutions using (6).

First, we examine the performance in the single-hop WLAN without non-PTP traffic applied. The two solutions (i.e., [13], [12]) using HW TS perform better than ours because they rely on a clock with finer resolution (i.e., a few ns in [12], 11.36 ns in [13]). As elaborated previously, we opt for the existing TSF in Wi-Fi as the PTP HW clock, which counts at 1 μs. Although this choice affects our performance, it is more hardware efficient and brings benefits to other applications

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

	HW TS	Extra HW clock	Sync inte-rval (sec)	Clock synchronization performance over single-hop wireless network						Clock synchronization performance across wired-wireless 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	-279 ns	820 ns	1.40 μs	-325 ns	868 ns	1.46 μs	866 ns	1.43 μs	2.81 μs	929 ns	1.42 μs	2.82 μs
[13]	√	√	0.1	0.24 ns	0.53 ns	**1.00** ns	NA	NA	NA	NA	NA	NA	NA	NA	NA
[12]	√	√	2.0	1.10 ns	1.76 ns	**3.40** ns	NA	NA	NA	NA	NA	NA	NA	NA	NA
[7]	×	×	2.0	-14.2 μs	27.7 μs	**51.2** μs	NA	NA	NA	NA	NA	NA	NA	NA	NA
[8]	×	×	1.0	6.60 μs	0.58 μs	**7.34** μs	NA	NA	NA	NA	NA	NA	NA	NA	NA
[12]	×	×	2.0	4.60 μs	1.58 μs	**6.63** μs	NA	NA	NA	NA	NA	NA	NA	NA	NA
[9]	×	×	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															
[25]	×	×	1.0	37.5 ms	6.80 μs	**37.6** ms	NA	NA	NA	NA	NA	NA	NA	NA	NA
[28]	×	×	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

relying on a synchronized TSF. A secondary factor is the *Sync interval*, [13] uses a much shorter synchronization period (i.e. 0.1 second), which means the solution makes a trade-off between performance and network overhead. Among the SW TS based solutions, [24] also outperforms ours in this condition. The authors of [24] has investigated the synchronization bias caused by the asymmetrical link delay measured when MCS used in the *Sync* and *Delay_request* packets are different. They have then carefully measured the delay in the Wi-Fi interface for PTP event packets, and use the measurement to calibrate the SW TS in the Wi-Fi driver. Although this solution achieved optimal performance for one Wi-Fi card, the calibration is hardware-dependent, and hence has to be performed for each Wi-Fi card and its corresponding driver. They also need to modify the PTP application to avoid using the default link delay measured on the fly. Our solution on the other hand uses standard PTP approach, it does not require pre-calibration and the performance is not subject to the used MCS in PTP event packets thanks to the nature of HW TS (i.e., TS happens at the start of frame).

Most solutions have not evaluated the impact of traffic load, and neither the performance beyond single-hop Wi-Fi network. The exceptions are [9], [24], [28]. [24] and [28] are the only solutions which provide 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 by network traffic. The CS error's σ increases from 360 ns to 1.26 μs in [24] and from 341 ns to 1.14 μs in [28].

[9] is the only solution other than ours that has evaluated CS performance across wired and wireless network segment. The CS performance of this solution is apparently better than ours in the absence of network load. The network topology of this work does resemble ours in the sense that it also has a GM in wired network, a Wi-Fi AP as a BC, and a Wi-Fi client as a PTP slave. However, further exploration unveils that the experiment setup employs expensive industrial PCs and dedicated NIC for PTP support. More particularly, they attach the *syn1588*[10] NIC on each of the devices (i.e., GM, AP and client), which costs more than 3000 USD per unit. The *syn1588* NIC offers an adder-based clock as the PTP hardware clock running on a 25 MHz oscillator. The clock is accessed by the customized driver of the Atheros 5212 Wi-Fi card to provide SW TS on the Wi-Fi interface. Hence the superior performance of [9] is thanks to the dedicated high quality clock with very fine resolution. The work has however used SW TS in the wireless part of the network and the performance of their work is not tested with non-PTP background traffic. Though the authors do mention that the solution is susceptible to non-PTP background traffic due to SW TS. Further, all the SW TS based solutions likely to suffer from propagation delay asymmetry due to different frame sizes and MCS [23].

To summarize, despite that some of the previous work do have good performance in specific settings, these solutions either rely on customized PTP stack, Wi-Fi stack or expensive and dedicated hardware, whereas our solution offers standard compliant PTP and Wi-Fi stack with no specific hardware requirement, making it more easily adopted in a real-life industrial network. In addition, our solution shows stable performance under network traffic, and it is the first to offer experimental analysis of PTP CS performance across Wi-Fi and Ethernet when traffic load is applied.

V. CONCLUSIONS

In this paper, a new approach for PTP with HW TS is proposed and verified over *openwifi*, an open-source IEEE 802.11 design on SDR. The *linuxptp* application and PHC subsystem of Linux kernel are employed as PTP software stack. Instead of adding a dedicated hardware clock, we aim to use the existing TSF timer of Wi-Fi standard as PTP HW clock, for the improved hardware-efficiency and the potentials of applications and Wi-Fi management layers relying on TSF for synchronization to benefit. The TSF timer however can only correct clock offset, hence modification is done to enable skew correction, making it qualified as a PTP HW clock. It is shown that P_{90} of our work is 1.40 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 on the clock synchronization accuracy is also investigated and proven to be insignificant. Lastly, the performance of our

[10]Syn1588 NIC https://www.oreganosystems.at/products/syn1588/hardware/syn1588r-pcie-nic

approach is tested across wired-wireless network and observed that the overall performance is stable and satisfactory. In the future, we consider to enhance the CS accuracy by tuning the hardware clock at finer granularity without compromising the hardware utilization. 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.

Our solution satisfies the performance requirement of *Wi-Fi Timesync* certificate, however it is not entirely 802.1AS compliant in the sense that we capture timestamps of regular PTP packet rather than the 802.11 Time Measurement packet, and we use a PTP boundary clock rather than transparent clock to form multi-hop PTP connections. The choice of this approach is mainly based on the maturity of available software and OS support. For future work, we are open to support 802.11 Time Measurement or Fine Time Measurement for clock synchronization when the relevant upper layer components are in place and when the performance requirement can not be reached by the current solution.

REFERENCES

[1] 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.

[2] *IEEE 1588-2019 - IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*, IEEE Std., 2019.

[3] 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.

[4] 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/

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

[6] D. Cavalcanti, S. Bush, M. Illouz, G. Kronauer, A. Regev, and G. Venkatesan, "Wireless tsn-definitions use cases & standards roadmap," *Avnu Alliance*, pp. 1–16, 2020.

[7] 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.

[8] 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.

[9] 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.

[10] 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*. IEEE, 2014, pp. 1–8.

[11] 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.

[12] 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.

[13] 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.

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

[15] 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

[16] 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

[17] S. Sudhakaran, V. Mageshkumar, A. Baxi, and D. Cavalcanti, "Enabling qos for collaborative robotics applications with wireless tsn," in *2021 IEEE International Conference on Communications Workshops (ICC Workshops)*. IEEE, 2021, pp. 1–6.

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

[19] 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.

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

[21] P. Chen and Z. Yang, "Understanding precision time protocol in today's wi-fi networks: A measurement study," in *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. USENIX Association, Jul. 2021, pp. 597–610. [Online]. Available: https://www.usenix.org/conference/atc21/presentation/chen

[22] Intel. dual band wireless ac 8260. [Online]. Available: https://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/dual-band-wireless-ac-8260-brief-v2.pdf

[23] 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.

[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] 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.

[26] 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.

[27] 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.

[28] 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.

[29] *802.11-2016 - IEEE Standard for Information technology—Telecommunications and information exchange between systems Local and metropolitan area networks—Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, IEEE Std., 2016.

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

[31] M. Chwalisz and A. Wolisz, "Towards efficient coexistence of ieee 802.15.4e tsch and ieee 802.11," in *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*, 2018, pp. 1–7.

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 (IEEE 802.15.4, IEEE 802.11), wireless time-sensitive networking and their real-time implementation on SoC 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 Full Professor at University of Antwerp, as well as 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 research interests are: 5G and Beyond architectures including edge computing; orchestration for flexible and programmable future end-to-end networks. He is also active in the domains of vehicular communications (cars, drones, vessels) and mobility. Prof. Marquez-Barja is co-leading the Citylab Smart City testbed, 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.

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 intelligent Wireless Networking (iWiNe) at Ghent University, where she is leading a team of more than 30 researchers. Ingrid Moerman is also Program Manager of the 'Deterministic Networking' track, part of the CONNECTIVITY program at imec, and in this role she coordinates research activities on end-to-end wired/wireless networking solutions driven by business- and mission-critical applications that have to meet strict Quality of Service requirements in terms of throughput, bounded latency and reliability in private wireless environments. Ingrid Moerman has a longstanding experience in running and coordinating national and EU research funded projects.