IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received December 9, 2020, accepted January 13, 2021, date of publication January 25, 2021, date of current version February 5, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3054164

Clock Synchronization Algorithms Over PTP-Unaware Networks: Reproducible Comparison Using an FPGA Testbed

IGOR FREIRE[1], (Member, IEEE), CAMILA NOVAES[1], IGOR ALMEIDA[2], (Member, IEEE), EDUARDO MEDEIROS[3], MIGUEL BERG[3], (Senior Member, IEEE), AND ALDEBARO KLAUTAU[1], (Senior Member, IEEE)

[1]LASSE-5G and IoT Research Group, Federal University of Pará, Belém 66075-750, Brazil
[2]Ericsson Research, Indaiatuba 13330-300, Brazil
[3]Ericsson Research, 164 80 Stockholm, Sweden

Corresponding author: Igor Freire (igorfreire@ufpa.br)

This work was supported in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior-Brasil (CAPES) under Grant Finance Code 001, and in part by the Innovation Center, Ericsson Telecomunicações S.A., Brazil.

ABSTRACT This work explores clock synchronization algorithms used to process timestamps from the IEEE 1588 precision time protocol (PTP). It focuses on the PTP-unaware network scenario, where the network nodes do not actively contribute to PTP's operation. This scenario typically imposes a harsh environment for accurate clock distribution, primarily due to the packet delay variation experienced by PTP packets. In this context, it is essential to process the noisy PTP measurements using algorithms and strategies that consider the underlying clock and packet delay models. This work surveys some attractive algorithms and introduces an open-source analysis library that combines several of them for better performance. It also provides an unprecedented comparison of the algorithms based on datasets acquired from a sophisticated testbed composed of field-programmable gate arrays (FPGAs). The investigation provides insights regarding the synchronization performance under various scenarios of background traffic and oscillator stability.

INDEX TERMS Clock synchronization, IEEE 1588, partial timing support, precision time protocol.

I. INTRODUCTION

The need for a global understanding of time among network elements and devices appears in a vast and growing range of modern applications. For example, it is essential to telecommunications [1], smart grids [2], data centers [3], industrial automation [4], financial applications [5], time-sensitive networks and distributed computing [6], and many more. The typical approach to provide a global time base to a node is to rely on the combination of global navigation satellite system (GNSS) disciplined clocks and packet-based distribution of synchronization over a network. For the latter, the network time protocol (NTP) and the IEEE 1588 precision time protocol (PTP) are the predominant standards.

NTP is prevalent in wide area networks and commonly used in applications that require relatively coarse synchronization in the order of milliseconds [7]. In contrast, PTP typically relies on more advanced hardware assistance and controlled network environments, where it achieves time accuracy in the order of nanoseconds. PTP has achieved widespread usage primarily since its 2008 revision, with the so-called PTPv2 [8]. It is continuously evolving and recently consolidated the IEEE 1588-2019 standard revision [9], which features, for example, the so-called High Accuracy profile for sub-nanosecond accuracy [10].

The key for high-accuracy timing transport through PTP is to rely on network elements that actively support the protocol, namely PTP-aware nodes. In particular, the so-called boundary clocks (BCs) and transparent clocks (TCs) [9]. The switches or routers containing BC or TC functionality provide special mechanisms to deal with the packet delay variation (PDV) that PTP messages experience when traversing a network. Thus, when all network components are PTP-aware, the PDV does not harm the synchronization accuracy. In the context of telecom networks, this scenario is called full timing support (FTS), where the PTP nodes operate with parameters

The associate editor coordinating the review of this manuscript and approving it for publication was Woorham Bae.

and protocol attributes (i.e., a PTP profile [9]) defined by ITU-T Recommendation G.8251.1 [11]. Such FTS networks also typically transport a reliable frequency reference over the physical layer (PHY) via synchronous Ethernet (SyncE) [12] or similar technology.

The opposite scenario is when some or all of the network elements are PTP-unaware, i.e., do not actively support PTP messages. In this case, the PDV and delay asymmetry affecting PTP messages can significantly degrade the synchronization performance. In the context of telecom networks, this scenario is called partial timing support (PTS), whose profile is defined in ITU-T Recommendation G.8275.2 [13]. It is often an attractive solution as it allows operators to transport PTP over their existing or third-party PTP-unaware networks.

There are two main use cases for PTS. The first is the distribution of PTP as a secondary source of time to back up GNSS, called assisted partial timing support (APTS). This is a compelling use case to protect from GNSS vulnerabilities to interference (intentional and unintentional) and weather-related antenna damage [14]. It is especially convenient when PTP can be distributed towards end applications that are already co-located with a GNSS-receiver, such as legacy code-division multiple access (CDMA) base stations [15].

The second main PTS use case is the distribution of PTP as the primary source of time in well-controlled networks with few hops [16]. For example, this is useful for the distribution of timing over third-party in-building or last-mile network segments connecting outdoor GNSS-disciplined clocks (e.g., on the roof) to indoor small cells or radio units [15].

In terms of performance, PTS is generally challenging and a less investigated topic. Only recently, the ITU-T has published Recommendation G.8273.4 [17], which specifies performance requirements for clocks in a PTS network.

The typical approach to achieve reasonable performance levels over a PTP-unaware network is to process the noisy measurements acquired via PTP timestamps. The timestamps reflect the PDV experienced by PTP packets and the random frequency and phase fluctuations experienced by the local oscillators. Thus, the processing algorithms can consider both phenomena. Also, in the APTS use case, the PTP processing can incorporate knowledge acquired while the device is locked to GNSS [14].

In this work, we explore a range of clock synchronization algorithms that can work well in PTP-unaware networks. More specifically, we discuss two main groups of estimators, both comprehensively surveyed in Section II. The first group refers to window-based packet selection and filtering algorithms, which process windows of metrics derived from PTP timestamps while focusing on overcoming the PDV. The second group consists of estimators based on least-squares (LS) and Kalman filtering (KF), which incorporate the oscillator's model as part of the estimation. The discussion is generic and applicable to any synchronization use case that can benefit from the simple and cost-effective distribution of timing over existing and well-controlled PTP-unaware networks. We provide several insights into the usage and rationale of the algorithms while expanding on the pre-existing literature.

Our primary goal is to provide a fair and reproducible comparison of the synchronization algorithms based on hardware acquisitions. The vast majority of the literature relies on simulation experiments or limited testbed setups. In this work, we exploit an advanced testbed based on field-programmable gate arrays (FPGAs), which collects nanosecond-accurate PTP timestamps from a real network and truth metrics for analysis. As discussed later, [18] presents a related testbed for comparison of synchronization algorithms. Our work follows a similar direction but uses significantly more accurate hardware meticulously designed for the experiments.

Furthermore, this work introduces the open-source PTP dataset analysis library (PTP-DAL),[1] developed in Python to process the datasets acquired from our testbed. With this library, we aim to enable reproducible results and foster advances in synchronization algorithms. This work describes PTP-DAL's processing architecture and particularly how it combines various algorithms for performance.

Our main contributions in this work include:

1) The survey, open-source implementation, and comparison of various PTP processing algorithms with insights regarding how to tune and combine them in practice, based on experiments with real hardware. While doing so, we contribute with the hardware-based evaluation of algorithms, such as KF, which, to the best of our knowledge, were previously discussed in the literature solely based on numerical simulations.
2) A thorough description of our FPGA-based testbed developed for PTP analysis. We discuss the main hardware components and FPGA design choices for the acquisition of datasets containing real PTP timestamps and the associated nanosecond-accurate truth metrics for reliable offline analysis.
3) A discussion regarding PTP delay distributions encountered on practical PTS network scenarios and a corresponding analysis of their impact on synchronization performance through testbed-based experiments.

This work is organized as follows. Section II presents a literature survey. Section III describes the model for time and frequency processes and measurements. Section IV formulates the algorithms that turn noisy measurements into more accurate time offset estimations. Section V describes the processing architecture that we adopt to evaluate and compare algorithms. Section VI describes the FPGA-based testbed used for data acquisition. Finally, Section VII presents experimental results and Section VIII concludes. For convenience, a list of acronyms and abbreviations follows.

[1] PTP-DAL is available online at https://github.com/lasseufpa/ptp-dal.

ACRONYMS

APTS	assisted partial timing support.
AWG	arbitrary waveform generator.
BC	boundary clock.
BG	background.
CBR	constant bit rate.
CDF	cumulative distribution function.
CDMA	code-division multiple access.
EAPF	earliest arrival packet filter.
FPGA	field-programmable gate array.
FTS	full timing support.
GbE	gigabit Ethernet.
GM	grandmaster.
GMII	gigabit media-independent interface.
GNSS	global navigation satellite system.
i.i.d	independent and identically distributed.
KF	Kalman filtering.
LO	local oscillator.
LP	linear programming.
LS	least-squares.
m-to-s	master-to-slave.
MAC	medium access control layer.
max|TE|	maximum absolute time error.
MSE	mean square error.
MVU	minimum variance unbiased.
NN	neural network.
NTP	network time protocol.
OCXO	oven-controlled crystal oscillator.
PDF	probability density function.
PDV	packet delay variation.
PHY	physical layer.
PLL	phase-locked loop.
PPS	pulse per second.
PRC	primary reference clock.
PTP	precision time protocol.
PTP-DAL	PTP dataset analysis library.
PTS	partial timing support.
RT	residence time.
RTC	real-time clock.
s-to-m	slave-to-master.
SD	standard deviation.
SyncE	synchronous Ethernet.
TC	transparent clock.
TE	time error.
ToD	time-of-day.
UTC	coordinated universal time.
VBR	variable bit rate.
VLAN	virtual local area network.
XO	crystal oscillator.
ZG	zero-gap.

II. RELATED WORK

In this section, we survey the literature regarding synchronization algorithms that can be robust to PDV in the context of PTP-unaware networks. As mentioned earlier, we discuss two main groups of time offset estimators: window-based processing and model-based filtering. Moreover, we briefly discuss the problem of delay asymmetry and some of the related literature. Lastly, we position this work relative to other synchronization testbeds.

A. WINDOW-BASED PROCESSING

The technique known as *packet selection* is an example of window-based processing, which has been extensively investigated in the literature. [19] analyzes three packet selection operators: the earliest arrival packet filter (EAPF), also known as sample-minimum, which selects packets with minimum delays in a window; the sample-mean, which averages a window of time offset measurements; and the sample-maximum, which is the opposite of EAPF. The analysis of [19] is backed by statistical and experimental characterization of the delays experienced by timing messages under two different network scenarios, referred to as *cross-traffic* and *in-line traffic*. It shows that, under cross-traffic, the delay is well characterized as an independent and identically distributed (i.i.d) random variable with Erlang probability density function (PDF), whose shape parameter relates to network load. In contrast, under in-line traffic through store-and-forward switches, the delay is characterized by a less tractable PDF that resembles a mirrored Erlang. More importantly, it emphasizes that the optimal selection operator varies. Under light cross-traffic load, the EAPF yields the best performance in terms of output noise variance. Under heavier cross-traffic load, in contrast, the Erlang delay distribution becomes closer to Gaussian, and so the sample-mean yields the best result. Finally, for in-line traffic, the sample-maximum operator outperforms the other two.

Based on [19], the work in [20] formulates a mechanism to adapt the packet selection operator in real-time. The approach concurrently filters the noisy delay measurements (i.e., the difference between arrival and departure timestamps) with the sample-minimum, sample-mean, and sample-maximum filters. At any moment, then, it chooses the output with minimum variance. However, a limitation of this approach is that it does not perceive the noise bias that in practice may differ significantly among the selection operators.

In a similar direction, [21] and [22] propose the *sample-mode* filtering technique, which distributes timestamp differences (arrival minus departure) collected during an observation window over bins and selects the mode. Effectively this is a selection operator that adapts to the delay distribution. However, the referred works do not discuss the more significant disadvantage of this operator, which is that performance highly depends on how narrow the distribution is around its theoretical mode or if a unique mode exists.

More generally, there are a few limitations to packet selection approaches. First and foremost, the selection operators only work well if the time offset remains reasonably constant within the observation window. Second, the conclusions in terms of best-performing operators in [19]–[22] are

restricted to specific network and delay models. Thus, empirical validation becomes necessary for a given network. Third, the approaches focus on the reduction of variance but are not guaranteed to yield unbiased estimates.

B. DELAY ASYMMETRY AND BIAS COMPENSATION

The problem of estimation bias exists in all time offset estimation techniques discussed in this work. Its primary cause is the asymmetry between the master-to-slave (m-to-s) and slave-to-master (s-to-m) end-to-end delays experienced by PTP messages. This asymmetry creates a bias on the two-way time offset measurements. Ultimately, it often represents one of the biggest hurdles to synchronization accuracy when transporting PTP over timing-unaware networks.

The literature discusses several sources of delay asymmetry originating from the timestamping implementation, the PHY hardware, and the delay components, i.e., propagation, processing, queuing, and transmission (or serialization) delays. At the hardware level, the transmit and receive paths of a network interface can impose different latencies from the point of packet arrival or departure to where timestamps are taken, as discussed in [23]. For example, this is typical in gigabit Ethernet (GbE) interfaces (i.e., 1000BASE-T), which we consider in this work. Unless known and calibrated, this latency difference leads to PTP delay asymmetry.

On the one hand, the PHY latency asymmetry effect is alleviated when the clock master and slave devices are paired with each other, as pointed out in [24]. On the other hand, over a PTP-unaware network, each switching stage between the PTP endpoints (master and slave) can also contribute with their asymmetric PHY latencies. This contribution from the network is harder to compensate for in practice, as it varies per-hop and potentially comes from third-party equipment.

At the link level, the propagation delay may be significantly asymmetric when the m-to-s and s-to-m transmissions occur over distinct wavelengths [25] or asymmetric transmission lines [24]. Also, the transmission delays of PTP messages can be asymmetric if the m-to-s and s-to-m link bit rates are distinct [26], [27], or if the master and slave nodes negotiate different line speeds with their peers [28].

Meanwhile, queuing delay asymmetry arises when the PTP traffic shares the network with background (BG) traffic. In this scenario, the queuing delays are typically asymmetric for each PTP exchange, and can also be asymmetric on average. For example, due to distinct BG traffic loads in the two directions [29], or asymmetric BG packet sizes.

The PTP messages also experience variable processing delays over the network. Such processing delay variations come primarily due to the access to shared switch resources when processing each packet [30]. Consequently, the two PTP messages of a two-way exchange can experience asymmetric processing delays. Nevertheless, unlike the queuing delays, the processing delays tend to be symmetric on average.

Lastly, network-level asymmetry arises when the m-to-s and s-to-m network paths are distinct [31]. For example, this is possible when PTP messages are transported over UDP, as in the case of the PTS PTP profile [13], given that the m-to-s and s-to-m routes can differ.

Table 1 summarizes the sources of delay asymmetry and the nature of each phenomenon in terms of whether it is static for all PTP packets or random. For example, the asymmetry that arises from propagation delay differences is typically static, whereas the asymmetry from queuing delays is typically stochastic. Nevertheless, note that the table shows the *typical* nature, which may not hold in all forms of synchronization and deployments. Furthermore, it focuses on the main asymmetry contribution of each effect. For example, while the PHY latency asymmetry can be variable, its primary contribution is usually static.

TABLE 1. Summary of delay asymmetry sources.

Source	Asymmetry On	Nature
PHY hardware	Tx/Rx Path Latencies	Static
Propagation Delay	Wavelengths or line lengths	Static
Transmission Delay	Link bit rates	Static
Queuing Delay	BG traffic	Stochastic
Processing Delay	Switch processing time	Stochastic
Network Routing	PTP message paths	Stochastic

The typical approaches for dealing with the delay asymmetry bias are estimation and calibration techniques. Furthermore, many works propose strategies to avoid asymmetry instead of correcting it. In terms of how one can achieve the asymmetry estimation, there are two main categories: methods that rely on extra probing packets (e.g., [32]), and methods that estimate the asymmetry based on the ordinary PTP timestamps (e.g., [29] and [33]).

The method proposed in [29] focuses on the asymmetry of queuing delays. It estimates the difference between each packet's queuing delay and the minimum observed queuing delay in a window. If the minimum queuing delays are symmetric in the m-to-s and s-to-m directions, these estimates can then be used to infer the queue-induced asymmetry disturbing each two-way PTP message exchange. This approach requires observation windows that are large enough to contain minimally delayed packets. Also, it requires symmetric minimum delays. Any residual static asymmetry on the minimum delays is left uncorrected.

The method proposed in [33] estimates the average asymmetry between gamma-distributed delays. Nevertheless, like packet selection algorithms, it requires that the time offset remains constant throughout observation windows. Besides, it relies on a few imperfect mathematical approximations, and it only works if the delays are indeed close to gamma-distributed. According to [19], this distribution is typical for cross-traffic scenarios but not for in-line traffic.

Lastly, an example strategy to avoid the delay asymmetry instead of estimating it is the so-called controlled departure method from [34]. The idea is to assign a sufficient gap between a PTP message and the preceding BG packet transmission, such that the PTP packet does not suffer queuing

delay over the network. With this approach, one can avoid queue-induced asymmetry (and most of the PDV). However, the method is limited to specific network topologies where PTP packets do not suffer contention. Although our testbed supports this technique, as demonstrated in [35], further evaluations using it are beyond the scope of this work.

C. MODEL-BASED FILTERING

Unlike packet selection, some estimation techniques incorporate models for time and frequency offsets. The two techniques of interest are the LS and KF algorithms. For example, [18] uses LS polynomial fitting on timestamp windows to model the slave time as a function of the master (reference) time. It aims at alleviating the noise from jittery timestamps taken in software. By using the LS-fitted time offset estimates, rather than the raw measurements, the output becomes less noisy, and the synchronization performance improves.

The model of [18] represents the slave time as a polynomial in the indeterminate t, where t is the absolute (reference) time. It admits a linear model, as well as contributions from quadratic and cubic terms, showing that the linear model is almost always the best to avoid overfitting and for less complexity. Moreover, given that the LS-fitting is applied on windows of timestamps, the paper analyzes how the window length impacts performance and particularly how the temperature conditions can affect the optimal window length. Nevertheless, it does not investigate the impact of the PDV.

A KF approach for clock synchronization is presented in [36]. Although formulated for an NTP application, it is equally valid for PTP. This work adopts a scalar-state vector-measurement KF formulation focusing on the discrepancies between intervals measured by the slave and the master. The state of the filter converges to the inter-departure interval of NTP packets sent from the slave to the master according to the slave's time base. By comparing this state to the nominal (or reference) inter-departure, the slave can infer its frequency offset. The method assumes a window of N messages during which the frequency offset remains constant. Also, it deals with the PDV by incorporating it as measurement noise. Since its measurement noise corresponds to the difference of consecutive message delays, the noise is a zero-mean process, but not necessarily Gaussian.

The primary investigation from [36] concerns the KF performance under two scenarios: white Gaussian noise and correlated noise. The latter represents the case of bursty BG cross-traffic that introduces self-similar queuing delays to PTP messages. In particular, it compares KF to a linear programming (LP) approach and simple moving average filtering. While KF outperformed the two other methods under Gaussian noise, LP was superior in the experiment with self-similar noise. Nevertheless, the evaluation is limited, given that it consists of a simulation where frequency offset remains constant throughout the experiment.

The LP method investigated in [36] consists of another model-based filtering alternative for clock synchronization. Its objective relates to the rationale of EAPF, i.e., selection of packets with minimum delays. The LP approach produces time offset estimates related to the minimum delays by minimizing the constraints of an LP problem.

The authors of [37] agree with the LP formulation from [36] and extend the experiments with more practical hardware impairments, such as oscillator noise and timestamping uncertainty. [37] also contrasts the LP and KF performances under Gaussian and self-similar delays, with the agreeing conclusions that LP can perform better than KF under non-Gaussian delay. Nevertheless, one disadvantage of LP is the relatively high computational cost for solving a new LP problem on every iteration. Besides, [36] and [37] consider delays in the order of milliseconds, such that their performance figures are poor compared to the nanosecond levels that we target in the remainder of this work.

The more recent work in [38] augments the LP approach of [37] with the compensation of the slave clock's frequency offset variations due to temperature. The goal is to linearize the slave clock's time offset before LP computations. Nevertheless, its experiments are based on numerical simulation, using a model for the relationship between the temperature and the oscillator's frequency offset, and assuming perfect knowledge of the temperature.

A more realistic KF evaluation is presented in [39]. Despite also using simulation, this work takes oscillator and timestamping uncertainties into account. In particular, it analyzes the accuracy of clock offset estimations obtained through a vector-state vector-measurement KF formulation under varying timestamping uncertainty levels, i.e., from hardware to software timestamping, and varying levels of oscillator stability. While neglecting packet delay uncertainties, it shows, for instance, that KF is helpful under poor (software) timestamping, but as the timestamping accuracy increases, the raw measurements approach the KF performance. Nevertheless, this conclusion only holds because the work ignores the delay uncertainty. Under high PDV, the latter becomes the predominant component of measurement noise, and so a Kalman filter can provide significantly smoother results than raw measurements, even with precise timestamping. However, [39] does not discuss this scenario.

One fallacy of KF for clock synchronization is that the algorithm is only optimal for Gaussian measurement and state noise, whereas in practical systems, this is not the case [40]. Another disadvantage comes from its relatively high computational cost. To this end, [41] proposes a more efficient KF approach. It considers a vector-state scalar-measurement formulation that avoids unnecessary Kalman iterations and, instead, can execute the *predict* and *update* steps of the filtering solely when necessary. Moreover, it expands the oscillator model from [39] and considers the effects of an unbiased delay compensation uncertainty.

An interesting discussion from [41] concerns the Kalman filter's sensitivity to its initialization parameters, particularly the measurement and process noise covariance matrices. The work shows how the estimation performance decreases when

the state noise covariance is wrongly initialized. In contrast, other investigations such as [39] evaluate the KF performance with covariances that match the noise introduced in simulation. In practice, it is difficult to have a priori knowledge about these covariances, so this becomes a disadvantage for the practical use of the KF approach.

Another noteworthy model-based estimator is the minimax estimator proposed in [42], which minimizes the maximum mean square of the normalized error corresponding to time and frequency offset estimates. This work focuses on the effects of PDV caused by BG traffic. Its optimal estimator assumes complete knowledge of the delay statistics and unlimited computational complexity, with a solution for imperfect delay knowledge based on [43]. However, it relies on the unrealistic assumptions that the time offset remains constant over a block of PTP exchanges (as discussed later, this requires perfect *syntonization*) and that the static delay asymmetry is known a priori. The work does not demonstrate the method's performance under real oscillator and networking conditions, as it relies on numerical simulations.

Lastly, another model-based time offset estimation strategy is the supervised learning approach in [44], which uses a neural network (NN) to improve the time offset fits obtained via LS. For each window of timestamps, the algorithm computes the LS linear regression, normalizes the LS fit, and feeds the normalized result into the NN. Then, the NN predicts the error between the LS fit and the true time offset at an arbitrary instant past the observation window, such that it can minimize the mean square error of the LS fit.

Similar to the investigation of [18], [44] evaluates the performance impact of the observation window length. The LS performance figures are convex with the window length, meaning the LS observation window cannot be too short and neither too large. In contrast, [44] shows that the NN tends to be more robust to oversized windows. This property may prove helpful in practice, as it could simplify the search for an optimal observation window length. However, it is not clear whether this result applies when the predominant noise is PDV. In [44] (and [18]), the focus is on the measurement noise due to temperature fluctuations, rather than PDV.

D. CLOCK SYNCHRONIZATION TESTBEDS

The majority of the literature regarding PTP estimation algorithms relies on numerical simulations. Inevitably, some works put significant effort into simulating oscillators and other uncertainties accurately. However, often such simulations do not entirely capture the real behavior of oscillators, timestamping, or networking conditions. Such a complete picture can only be provided by analyzing data from real hardware. Table 2 categorizes the works referred thus far in terms of their evaluation environment, whether based on numerical simulation or data from real hardware.

An essential feature for hardware-based evaluation of PTP processing algorithms is the ability to assess the accuracy of estimates. While a simulation environment always holds, e.g., the true time offset of a clock at any simulation instant, on a testbed, this requires extra effort. Another challenge for hardware-based evaluation is the ability to run multiple algorithms based on the same data so that algorithms can be compared under the same environmental conditions.

TABLE 2. Categorization of the referenced literature in terms of their experimental environment: simulation or hardware.

Evaluation	References
Simulation	[20]–[22], [29], [31]–[33], [36]–[39], [41]–[43]
Hardware	[18], [19] (packet delay traces only), and [44]

Such pre-requisites have been addressed in [18] and [44]. These works aim at a reliable comparison of algorithms by using the testbed of [40], which supports the acquisition of the timestamps exchanged by the synchronization protocol and the true synchronization error between the slave and master devices throughout the experiment. However, the testbed in [40] relies on relatively limited hardware. In particular, it uses software-based timestamps and provides accuracy and precision within the microsecond range. In contrast, the testbed that we present in this work is based on nanosecond-accurate and hardware-based timestamps, and it also addresses the requirements for reliable algorithm comparison.

Furthermore, to the best of our knowledge, the testbed from [40] has only been used to evaluate the LS (in [18]) and NN (in [44]) estimators. In this work, we explore a more extensive range of algorithms, including the strategies discussed in [20]–[22], [39], [41] and [18].

III. SYSTEM MODEL AND DEFINITIONS

In this section, we discuss the main definitions and models regarding the time and frequency offset processes. Furthermore, we describe how these processes can be measured using timestamps from PTP messages.

A. NOTATION

In this work, x and y represent time and frequency offset, respectively. $x[n]$ expresses the n-th true time offset value, whereas $\tilde{x}[n]$ denotes the n-th measurement of $x[n]$, which is corrupted by noise and subject to imperfections. More generally, $\tilde{z}$ represents a noisy measurement of z (a generic variable), whereas $\hat{z}$ denotes an elaborate estimate of z.

In terms of indexing, n indexes individual samples in discrete-time domain. In contrast, index k represents windows (i.e., collections) of samples. For example, $\hat{x}[k]$ denotes a time offset estimate based on the k-th collection of noisy measurements. When focusing on windows, we index some variables based on their position m in the window and the window index k, such as the version of the time offset $x[n]$ that is denoted as $x[k, m]$. The non-windowed and windowed domains can be reconciled by setting the non-windowed index n equal to $kN + m$. This identity holds provided that the observation windows are non-overlapping and of size N.

Lastly, bold lowercase letters such as **x** represent vectors, bold uppercase letters such as **H** represent matrices, and $(.)^T$ denotes the transposition operation.

B. TERMINOLOGY

Before delving into further definitions, it is worth clarifying the terminology regarding the overused word *clock*. More specifically, the distinction between a *clock* and a *clock signal* in this work. The term *clock* refers to a time counter, such as a real-time clock (RTC), or a device containing one. For example, in the IEEE 1588 standard [9], a *clock* means a network node that participates in PTP (i.e., exchanges messages of the protocol) and can measure the passage of time. In contrast, we use the term *clock signal* to refer to an analog periodic signal that drives the actions of the digital circuit that implements the RTC. This is the analog signal derived from a local oscillator (LO). The terms *clock frequency* and *clock cycle* refer to the frequency and period of the *clock signal*.

C. TIME AND FREQUENCY OFFSET PROCESSES

An RTC is a hardware structure capable of measuring real-time. It often holds a full time-of-day (ToD) count tracing an internationally-recognized time standard, such as coordinated universal time (UTC). For example, PTP RTCs typically use 32 bits to store nanoseconds and 48 bits to store seconds elapsed since January 1st, 1970 [9].

The RTC's implementation generally consists of a counter that operates with a nominal *clock frequency* of f_{nom}. On each clock cycle, the RTC increments its time count by $1/f_{\text{nom}}$. However, in practice, the frequency driving the RTC differs from nominal, and, as a result, the RTC increments its time count by an imperfect interval on every clock cycle. In this process, the RTC continuously accumulates time offset.

As explained in ITU-T Recommendation G.810 [45, Appendix I], the error between the time of a local clock and the time of a reference clock can be modeled as:

$$x(t) = x_0 + y_0 t + \frac{D}{2}t^2 + \frac{\phi(t)}{2\pi f_{\text{nom}}}, \tag{1}$$

where x_0 is the initial time offset, y_0 is the fractional frequency offset relative to the nominal frequency, D is the linear fractional frequency drift rate, and $\phi(t)$ is the phase noise. In this model, x_0, y_0, and D are deterministic parameters. In contrast, $\phi(t)$ models the random phase deviations of the *clock signal* that drives the RTC and is responsible for all random variations of $x(t)$.

The corresponding continuous-time frequency offset is the derivative of the time offset, given by:

$$y(t) = y_0 + Dt + \frac{1}{2\pi f_{\text{nom}}}\frac{d\phi(t)}{dt}. \tag{2}$$

Just like the time offset, the frequency offset changes continuously due to oscillator instabilities modeled by the frequency drift D and the random phase noise $\phi(t)$ [46].

Instead of pursuing the complete evolution of the continuous-time $x(t)$, many estimation algorithms focus on sufficiently short observation windows. The primary justification for this choice is the simplification of the model. First, the linear frequency drift D from (1) is generally negligible in the short-term, as it models long-term frequency deviations due to oscillator aging. Secondly, the phase noise varies relatively slowly, so that a sufficiently short observation window may capture a constant phase noise contribution to time and frequency offsets. In the end, the model becomes a piecewise linear approximation of (1), as discussed, e.g., in [36], [37].

In discrete-time, the simplified short-term piecewise linear model for a window of N samples of $x(t)$ is given by:

$$x[k,m] = x_0[k] + y[k]\tau[k,m], \quad 0 \le m < N, k \ge 0 \tag{3}$$

where $x_0[k]$ and $y[k]$ are the initial time offset and the normalized (or fractional) frequency offset of the k-th observation window, respectively, both assumed constant throughout the window. Also, this model assumes that the time offset samples are taken at irregular instants, rather than periodically, and so variable $\tau[k,m]$ denotes the true instant of each sample. For simplicity, this instant is measured relative to the beginning of the observation interval, such that $\tau[k,0] = 0$.

D. TIME OFFSET MEASUREMENT

The typical approach for time offset measurement is to execute a two-way exchange of timestamped messages between two clocks. For example, the so-called delay request-response mechanism from PTP [9], which is illustrated in Fig. 1. On the n-th iteration, the PTP master clock sends the Sync message, whose departure timestamp is $t_1[n]$ and whose arrival time at the PTP slave clock is timestamped as $t_2[n]$. The slave replies with a DelayReq message, which departs at time $t_3[n]$ and arrives back to the master at $t_4[n]$.

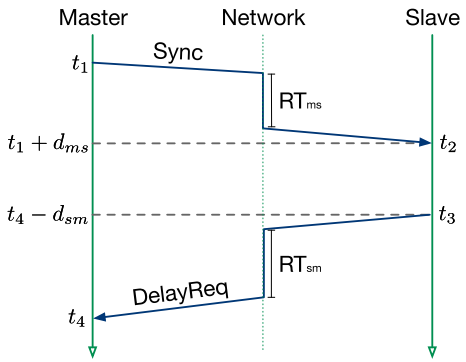


FIGURE 1. PTP delay request-response exchange with asymmetric queuing delays over the network.

Timestamps $t_2[n]$ and $t_3[n]$ are taken at the PTP slave side and, therefore, are corrupted by the slave's time offset relative to the reference time, i.e., the PTP grandmaster (GM) time. Also, the two messages experience distinct transit delays. The n-th Sync message experiences the m-to-s delay $d_{ms}[n]$, and the n-th DelayReq experiences the s-to-m delay $d_{sm}[n]$.

Consequently, the n-th time offset $x[n]$ can be expressed by:

$$\begin{cases} x[n] = t_2[n] - (t_1[n] + d_{ms}[n]) \\ x[n] = t_3[n] - (t_4[n] - d_{sm}[n]), \end{cases} \tag{4}$$

where $x[n]$ is assumed constant throughout the message exchange. Note that both equations compute the time offset as the slave time (t_2 or t_3) minus the master time (t_1 or t_4), adjusted by delays. More specifically, they refer to the instants highlighted by the dashed lines in Fig. 1.

The slave clock only observes the timestamps and does not know the true one-way delays of each message. In particular, the slave can compute the following timestamp differences:

$$\begin{cases} t_{21}[n] = t_2[n] - t_1[n] = x[n] + d_{ms}[n] \\ t_{43}[n] = t_4[n] - t_3[n] = -x[n] + d_{sm}[n], \end{cases} \tag{5}$$

which consist of two linearly independent equations with three unknowns: $x[n]$, $d_{ms}[n]$, and $d_{sm}[n]$. Then, the typical approach for time offset estimation is to assume symmetric delays and use the following linear combination of (5):

$$\tilde{x}[n] = \frac{t_{21}[n] - t_{43}[n]}{2}. \tag{6}$$

By substituting (5) in (6), note that (6) in reality yields noisy time offset measurements due to the delay asymmetry $w[n]$:

$$\tilde{x}[n] = x[n] + w[n] = x[n] + \frac{d_{ms}[n] - d_{sm}[n]}{2}. \tag{7}$$

In practice, even if the forward and reverse one-way delays were perfectly symmetric, there would still be other uncertainties on the timestamps that would make the measurements from (6) noisy. For example, there are uncertainties due to timestamp quantization and short-term oscillator stability. Nevertheless, we assume that the delay asymmetry is by far the predominant measurement noise component.

An alternative linear combination of (5) yields the so-called two-way delay measurement, namely the average between the m-to-s and s-to-m one-way delays:

$$\tilde{d}[n] = \frac{t_{21}[n] + t_{43}[n]}{2} = \frac{d_{ms}[n] + d_{sm}[n]}{2}. \tag{8}$$

Nevertheless, this estimate does not reveal the individual one-way delays, which are of interest.

The one-way delays are composed of fixed and random parts, both of which can be asymmetric, as follows:

$$\begin{cases} d_{ms}[n] = \kappa_{ms} + \delta_{ms}[n] \\ d_{sm}[n] = \kappa_{sm} + \delta_{sm}[n]. \end{cases} \tag{9}$$

Thus, the delay asymmetry noise $w[n]$ in (7) has both static and random asymmetry components, that is:

$$w[n] = \frac{(\kappa_{ms} - \kappa_{sm})}{2} + \frac{(\delta_{ms}[n] - \delta_{sm}[n])}{2}. \tag{10}$$

In this work, among the asymmetry sources in Table 1, we assume that the static PHY hardware and the average queuing delay asymmetry are the phenomena that contribute to the static asymmetry $(\kappa_{ms} - \kappa_{sm})/2$. Furthermore, we assume that the random processing and queuing delays of each PTP exchange determine the instantaneous asymmetry between $\delta_{ms}[n]$ and $\delta_{sm}[n]$, typically with a far more significant contribution from queuing delays. We neglect the other sources from Table 1 by assuming that bit rates, transmission line lengths, and network paths are symmetric.

A PTP-aware network overcomes the noise $w[n]$ by either handling the synchronization on each link with the so-called BCs or by measuring the full residence time (RT) of a PTP message over all PTP TCs [9] in the network. For example, RT measurements (RT_{ms} and RT_{sm}) are illustrated in Fig. 1. In contrast, when PTP runs end-to-end over a PTP-unaware network, there is no mechanism to overcome the noise $w[n]$, which then becomes a significant impairment.

In the end, there are two main challenges to be addressed by synchronization algorithms on the PTP-unaware network scenario. The first is the measurement noise $w[n]$. The second is the ability to trace the true time offset from (1), which evolves randomly over time.

E. FREQUENCY OFFSET MEASUREMENT

Because time synchronization implies frequency synchronization (i.e., *syntonization*), if the time offset measurements are accurate and frequent enough, then frequency offset measurements can be unnecessary in many applications. On the other hand, when the time offset measurements are not accurate, the frequency offset correction becomes essential to support the time offset estimation.

In PTP-unaware networks, the syntonization relies on measurements of the frequency offset based on PTP timestamps. However, one challenge is that frequency offset measurements depend on observation intervals rather than instantaneous observations. The usual approach is to consider the discrete-time approximation of (2), given by:

$$y[n] \approx \frac{x[n] - x[n-N]}{t_1[n] - t_1[n-N]}, \tag{11}$$

where $t_1[n] - t_1[n-N]$ is the time interval between $x[n-N]$ and $x[n]$, according to timestamps taken by the master clock, and N represents the observation window. This approximation only holds if the frequency offset remains constant during the observation interval controlled by N.

The typical way to measure the approximated frequency offset from (11) uses the measurements from (6), as follows:

$$\tilde{y}[n] = \frac{\tilde{x}[n] - \tilde{x}[n-N]}{t_1[n] - t_1[n-N]}. \tag{12}$$

Using (7), note this measurement is equivalent to:

$$\tilde{y}[n] = y[n] + \frac{w[n] - w[n-N]}{t_1[n] - t_1[n-N]}, \tag{13}$$

which, unlike (7), is an unbiased estimate.

Similar unbiased estimators can be derived based on timestamps from PTP messages in a single direction. For example, using only the m-to-s timestamps, one can use:

$$\tilde{y}[n] = \frac{t_{21}[n] - t_{21}[n-N]}{t_1[n] - t_1[n-N]}, \tag{14}$$

which, based on (5), is equivalent to:

$$\tilde{y}[n] = y[n] + \frac{d_{ms}[n] - d_{ms}[n-N]}{t_1[n] - t_1[n-N]}. \tag{15}$$

The difference between the two-way computation in (12) and the one-way computation in (14) is the noise term. The two-way approach balances the noise contributions from the PDV in the m-to-s and s-to-m directions. In contrast, (14) experiences the PDV in the m-to-s direction only.

In (13) and (15), note that the observation window N determines interval $t_1[n] - t_1[n-N]$ and, consequently, the noise attenuation. Nevertheless, it takes a long interval to attenuate the expected noise range. For example, if the delay fluctuations are in the order of microseconds, the interval $t_1[n] - t_1[n-N]$ has to be in the order of 1000 seconds to reduce the estimation noise to sub-ppb range. The problem is that the approximation in (11) also assumes that the frequency offset remains constant over the window of N samples. This is a conflicting requirement, which requires the observation window to be short enough.

IV. SYNCHRONIZATION ALGORITHMS

The goal going forward is to combine multiple *measurements* $\tilde{x}[n]$ from (6) to obtain less noisy time offset *estimates*. For the methods based on observation windows, we take N observations and output a single estimate. Also, if each window overlaps with $N-1$ samples of the previous window, one estimate can be obtained for each new $\tilde{x}[n]$ measurement. In this case, the estimator's output rate is the same as its measurement input rate despite the window-based approach. However, to simplify the notation, we assume non-overlapping windows in the sequel.

A. CONSTANT-OFFSET WINDOW PROCESSING

We start with a class of estimators that relies on the assumption that the time-offset remains constant throughout each observation window. For example, the estimators referred to as *packet selection* in the literature [19]–[22], [47], which select a pair of PTP messages in a window (one in each direction), then estimate the time offset using this pair only. Also, the strategies that do not strictly use selection, but which process a window of measurements using the constant time offset assumption. For example, the so-called *sample-average* (or *sample-mean*) and *sample-mode* approaches.

This class of estimators generally processes the timestamps differences $t_{21}[n]$ and $t_{43}[n]$ from (5). That is, in the k-th observation window, they process the vectors:

$$\mathbf{t_{21}}[k] = [t_{21}[k,0], t_{21}[k,1], \cdots, t_{21}[k,N-1]]^T \tag{16}$$
$$\mathbf{t_{43}}[k] = [t_{43}[k,0], t_{43}[k,1], \cdots, t_{43}[k,N-1]]^T, \tag{17}$$

each containing N timestamp differences.

As discussed in [48], the final estimate is obtained by:

$$\hat{x}[k] = \frac{\xi\{\mathbf{t_{21}}[k]\} - \xi\{\mathbf{t_{43}}[k]\}}{2}, \tag{18}$$

where $\xi\{\}$ denotes an arbitrary operator, typically the minimum, maximum, mean, median, or mode[2] operator.

Using (5), note that, if the time offset $x[kN+m]$ is constant over the k-th observation window (for $0 \leq m < N$), it can be factored out of the operator, so that (18) becomes:

$$\hat{x}[k] \approx x[kN] + \frac{\xi\{\mathbf{d_{ms}}[k]\} - \xi\{\mathbf{d_{sm}}[k]\}}{2}, \tag{19}$$

where $x[kN]$ denotes the time offset of the k-th window, constant within the index range $[kN, (k+1)N)$, while $\mathbf{d_{ms}}[k]$ and $\mathbf{d_{sm}}[k]$ are vectors given by:

$$\mathbf{d_{ms}}[k] = [d_{ms}[k,0], \cdots, d_{ms}[k,N-1]]^T \tag{20}$$
$$\mathbf{d_{sm}}[k] = [d_{sm}[k,0], \cdots, d_{sm}[k,N-1]]^T. \tag{21}$$

From (19), note the goal is to maximize the chances of having $\xi\{\mathbf{d_{ms}}[k]\}$ equal to $\xi\{\mathbf{d_{sm}}[k]\}$ within N measurements, such that these terms can cancel each other. This approach's success depends not only on the statistic pursued by the operator being symmetric (condition 1) but also on the chances of finding such symmetric realizations within N measurements (condition 2). For instance, when PTP shares the network with BG traffic, some PTP messages may still be lucky enough to traverse the entire network without colliding with BG packets, i.e., with no queuing delay. In this case, if all asymmetry sources from Table 1 other than queuing delay are absent, it is theoretically possible that the minimum $t_{21}[n]$ and $t_{43}[n]$ in N realizations becomes symmetric. Furthermore, if the delay distributions are more concentrated around their minima, the referred lucky realizations are more likely within N samples, as discussed in [19].

Due to its probabilistic nature, the algorithm's performance depends strongly on the window length. By increasing N, the chances of finding symmetric realizations can increase. On the other hand, note that (19) assumes that the time offset remains constant throughout each observation window. This is a vital requirement of the method, which effectively limits the window length. That is, N must be low enough such that $x[n]$ remains reasonably constant over N samples.

The time offset can only be constant over an observation window if the slave's RTC is perfectly syntonized. However, accurate syntonization is often a challenge in PTP-unaware networks. The syntonization is handled based on PTP estimates instead of, e.g., a PHY frequency reference. Thus, the process is prone to PDV-induced estimation errors and commonly low responsiveness to frequency deviations.

In this scenario, it is useful to apply a further syntonization layer within the software-level when executing the window-based algorithm. The proposed adaptation of (18) is to precede its computation with a time offset drift compensation step. More specifically, we can correct the values of $t_{21}[n]$ and $t_{43}[n]$ as follows:

$$\begin{cases} t'_{21}[k,m] = t_{21}[k,m] - \sum_{j=0}^{m} \hat{\Delta}_x[j] \\ t'_{43}[k,m] = t_{43}[k,m] + \sum_{j=0}^{m} \hat{\Delta}_x[j], \end{cases} \tag{22}$$

[2]The mode operator is typically preceded by the quantization of $t_{21}[n]$ and $t_{43}[n]$ and succeeded by the dequantization of the operator's results.

where $\hat{\Delta}_x[n]$ represents an estimate of the time offset drift $\Delta_x[n] = x[n] - x[n-1]$ (discussed in Section IV-D).

The resulting drift-compensated timestamp differences from (22) can be organized into vectors $\mathbf{t'_{21}}[k]$ and $\mathbf{t'_{43}}[k]$, as in (16) and (17). Then, similarly to (18), these vectors are used to estimate the time offset as follows:

$$\hat{x}'[k] = \frac{\xi\left\{\mathbf{t'_{21}}[k]\right\} - \xi\left\{\mathbf{t'_{43}}[k]\right\}}{2} + \sum_{j=0}^{N-1} \hat{\Delta}_x[j]. \quad (23)$$

where the summation term reintroduces the drift subtracted from the timestamp difference samples, such that $\hat{x}'[k]$ estimates the time offset by the end of the observation window.

The advantage of the estimator implemented by (23) is that it tolerates time-varying time and frequency offsets over the observation window. In this case, the window length is limited by the accuracy of the drift estimates $\hat{\Delta}_x[n]$, which tends to be a more relaxed constraint than the requirement of a constant time offset throughout the observation windows. As a result, this approach tends to perform well with long observation windows, as discussed later in Section VII. To the best of our knowledge, this formulation in (23) has not been discussed previously in the literature.

B. LEAST-SQUARES ESTIMATOR

Unlike the previous estimator, the LS estimator completely neglects the statistics of the noise $w[n]$. Instead, it focuses on the properties of the time offset process. More specifically, it assumes that the time offset is piecewise linear, as in the simplified model of (3). Then, it tries to find the LS fit of the linear model based on the measurements.

First, let us convert the non-windowed time offset measurements of (7) into windowed notation, as follows:

$$\tilde{x}[k, m] = x[k, m] + w[k, m]. \quad 0 \le m < N, k \ge 0 \quad (24)$$

Next, using (3), we model the measurements as follows:

$$\tilde{x}[k, m] = x_0[k] + y[k]\tau[k, m] + w[k, m]. \quad (25)$$

In vectorized form, we can define:

$$\tilde{\mathbf{x}}[k] = \left[\tilde{x}[k, 0], \tilde{x}[k, 1], \cdots, \tilde{x}[k, N-1]\right]^T \quad (26)$$

$$\mathbf{w}[k] = [w[k, 0], w[k, 1], \cdots, w[k, N-1]]^T \quad (27)$$

$$\boldsymbol{\theta}[k] = [x_0[k], y[k]]^T \quad (28)$$

$$\mathbf{H}[k] = \begin{bmatrix} 1 & \tau[k, 0] \\ 1 & \tau[k, 1] \\ \vdots & \vdots \\ 1 & \tau[k, N-1] \end{bmatrix}. \quad (29)$$

Thus, it follows that:

$$\tilde{\mathbf{x}}[k] = \mathbf{H}[k]\boldsymbol{\theta}[k] + \mathbf{w}[k], \quad (30)$$

where $\tilde{\mathbf{x}}[k]$ is the observed data vector, $\mathbf{H}[k]$ is the so-called observation matrix, $\boldsymbol{\theta}[k]$ is the vector of unknowns, and $\mathbf{w}[k]$ is the noise vector.

If the noise was zero-mean, white and Gaussian-distributed, it is known that the minimum variance unbiased (MVU) estimator of $\boldsymbol{\theta}[k]$ would be achieved by the LS estimate, which follows:

$$\hat{\boldsymbol{\theta}}[k] = \left(\mathbf{H}^T[k]\mathbf{H}[k]\right)^{-1} \mathbf{H}^T[k]\tilde{\mathbf{x}}[k]. \quad (31)$$

Nevertheless, since the delays that contribute to $w[n]$ are not white Gaussian in practice, the LS estimator's optimality is not guaranteed. Also, the LS estimates are subject to the inaccuracies of the piecewise linear model of (3) relative to the non-linear, stochastic, and continuous-time model of (1).

With the estimate $\hat{\boldsymbol{\theta}}[k]$, it is possible to compute the LS fit of the time offset as a line with slope $\hat{y}[k]$ starting from $\hat{x}_0[k]$:

$$\hat{x}[k, m] \doteq \hat{x}_0[k] + \hat{y}[k]\tau[k, m]. \quad 0 \le m < N, k \ge 0 \quad (32)$$

However, note that $\tau[k, m]$ represents the instant of the m-th observation (or PTP exchange) relative to the beginning of the window. This instant is unknown, but it can be approximated based on timestamps taken on the GM side (e.g., timestamp t_1). Alternatively, one can assume that the observations are periodic with period T (the PTP exchange interval), such that the observation matrix simplifies to:

$$\mathbf{H}[k] = \begin{bmatrix} 1 & 0 \\ 1 & T \\ \vdots & \vdots \\ 1 & (N-1)T \end{bmatrix}. \quad (33)$$

Accordingly, the LS fit becomes:

$$\hat{x}[k, m] \doteq \hat{x}_0[k] + \hat{y}[k](mT). \quad 0 \le m < N, k \ge 0 \quad (34)$$

In practice, overlapping windows can be used such that an observation window has a single new measurement and $N-1$ measurements from the previous window. In this case, the output of the LS estimator can be solely the last fitted value from (32) or (34), i.e., $\hat{x}[k, N-1]$, so that there is one output for every input $\tilde{x}[n]$.

Importantly, note that, in contrast to the algorithms from Section IV-A, the LS estimator does not require constant time offsets throughout the observation windows. Instead, it requires the frequency offset to remain constant, such that the linear time offset model holds. Thus, the LS window length is limited by the frequency stability. Besides, note that quadratic, cubic, and other terms could be introduced in the LS formulation. However, as discussed in [18], the piecewise linear model tends to produce better results.

C. KALMAN FILTERING

The KF approach departs from window-based strategies and particularly from the assumption that the unknown time and frequency offsets are constant (or deterministic) within the observation windows. Instead, it considers that $x[n]$ and $y[n]$ are continuously-evolving stochastic processes that compose a *state vector* $\boldsymbol{s}[n] = [x[n], y[n]]^T$. The goal of the filter is to predict the state $\boldsymbol{s}[n]$ with minimum mean square error (MSE) based on past observations and knowledge of the dynamics of the time and frequency offset processes.

In the conventional KF formulation, the state vector $\boldsymbol{s}[n]$ consists of a first-order Gauss-Markov process [49]. Thus, we start by establishing recursive models for $x[n]$ and $y[n]$, such that we can express $\boldsymbol{s}[n]$ based on $\boldsymbol{s}[n-1]$.

The recursive model of the time offset is given by:

$$x[n] = x[n-1] + y[n-1]\Delta\tau[n] + u_x[n]. \tag{35}$$

This model assumes that, from $x[n-1]$ to $x[n]$, the main change comes from the linear contribution of the frequency offset $y[n-1]$, which acts over the interval $\Delta\tau[n] = \tau[n] - \tau[n-1]$. This part of the model relates to the piecewise linear model of (3). However, (35) assumes that the frequency offset is constant only within the short interval $\Delta\tau[n]$, rather than over a long observation window. Furthermore, (35) acknowledges that the linear term does not capture all the complexity of the continuous-time model in (1). In particular, it includes an uncertainty term $u_x[n]$, which comes primarily due to the contribution from the phase noise $\phi(t)$ in (1).

Next, the recursive frequency offset model is given by:

$$y[n] = y[n-1] + u_y[n], \tag{36}$$

where $u_y[n]$ represents the frequency state noise, which is responsible for time variations of the frequency offset.

Using (35) and (36), and given that $\boldsymbol{s}[n] = [x[n], y[n]]^T$, note that the Gauss-Markov state process can be modeled by:

$$\boldsymbol{s}[n] = \mathbf{A}\boldsymbol{s}[n-1] + \mathbf{u}[n], \quad n \geq 0 \tag{37}$$

where $\mathbf{A}$ is the *state transition matrix* given by:

$$\mathbf{A} = \begin{bmatrix} 1 & T \\ 0 & 1 \end{bmatrix}, \tag{38}$$

and $\mathbf{u}[n] = \left[u_x[n], u_y[n]\right]^T$ is the *state noise vector*. For simplicity, as in (33), this transition matrix assumes that the sample intervals $\Delta\tau[n]$ in (35) are constant with period T.

In the conventional KF formulation, $\mathbf{u}[n]$ in (37) consists of a zero-mean white Gaussian vector with PDF $\sim \mathcal{N}(\mathbf{0}, \mathbf{Q})$. Matrix $\mathbf{Q} = E\left[\mathbf{u}[m]\mathbf{u}^T[n]\right]$ is the *state noise covariance matrix*, and it is assumed that $\mathbf{Q} = 0$ for $m \neq n$, namely that the noise vectors are independent from sample instant to sample instant. Furthermore, the formulation assumes that the initial state $\boldsymbol{s}[-1]$ is a random vector independent of $\mathbf{u}[n]$ and whose PDF is $\sim \mathcal{N}(\mu_s, \mathbf{C}_s)$. Here, we assume also that $u_x[n]$ and $u_y[n]$ are uncorrelated so that $\mathbf{Q}$ is a diagonal matrix for all $m = n$, as assumed in [39], [41].

The statistics assumed for the noise vector $\mathbf{u}[n]$ imply that both time and frequency offsets experience random-walk, i.e., they continuously accumulate a sequence of i.i.d Gaussian random variables. In terms of frequency stability characterization [46], [50], this model considers white frequency noise (which causes random-walk in time) and random-walk frequency noise. As mentioned in [39], these tend to be the predominant components of phase noise, and the advantage here is that they lead to the more tractable case of Gaussian state noise. However, other neglected phase noise sources (particularly flicker phase, flicker frequency, and white phase noise) may also be relevant in practice.

Next, we define the measurement model. In this work, we opt for a vector-state scalar-observation KF configuration [49], similar to [41], where the observations are the time offset measurements from (6). Hence, the model becomes:

$$\tilde{x}[n] = \mathbf{h}^T\boldsymbol{s}[n] + w[n], \tag{39}$$

where $\mathbf{h} = [1, 0]^T$ is a (2×1) vector such that this equation is equivalent to (7).

Similarly to the state noise, the conventional KF formulation assumes that the scalar measurement noise is zero-mean Gaussian-distributed with uncorrelated samples, variance σ_w^2, independent to both the initial state $\boldsymbol{s}[-1]$ and the state noise vector $\mathbf{u}[n]$. Here, in particular, the Gaussian assumption does not hold, given that the noise in (10) is nearly always non-Gaussian. Hence, the given Kalman filter is not guaranteed to be optimal [36], just like the LS estimator of Section IV-B is not guaranteed to be the MVU estimator. Nevertheless, both estimators are still of interest. The Kalman filter is still the optimal linear minimum MSE estimator [49].

Finally, the conventional vector-state scalar-measurement KF equations can be used (see, e.g., [49]). For the given model, the sequence of equations from the so-called *prediction* to *correction* (or *update*) steps of the filtering are:

$$\hat{\boldsymbol{s}}[n|n-1] = \mathbf{A}\hat{\boldsymbol{s}}[n-1|n-1] \tag{40}$$

$$\mathbf{M}[n|n-1] = \mathbf{A}\mathbf{M}[n-1|n-1]\mathbf{A}^T + \mathbf{Q} \tag{41}$$

$$\mathbf{k}[n] = \frac{\mathbf{M}[n|n-1]\mathbf{h}}{\sigma_w^2 + \mathbf{h}^T\mathbf{M}[n|n-1]\mathbf{h}} \tag{42}$$

$$\hat{\boldsymbol{s}}[n|n] = \hat{\boldsymbol{s}}[n|n-1] + \mathbf{k}[n]\left(\tilde{x}[n] + \mathbf{h}^T\hat{\boldsymbol{s}}[n|n-1]\right) \tag{43}$$

$$\mathbf{M}[n|n] = \left(\mathbf{I} - \mathbf{k}[n]\mathbf{h}^T\right)\mathbf{M}[n|n-1], \tag{44}$$

where $\hat{\boldsymbol{s}}[l|p]$ is the estimate of state $\boldsymbol{s}[l]$ based on the p-th state, $\mathbf{M}[l|p]$ is the corresponding state estimate covariance matrix, defined as $E\left[\left(\boldsymbol{s}[l] - \hat{\boldsymbol{s}}[l|p]\right)\left(\boldsymbol{s}[l] - \hat{\boldsymbol{s}}[l|p]\right)^T\right]$, $\mathbf{k}[n]$ is the 2×1 Kalman gain vector, and $\mathbf{I}$ is a 2×2 identity matrix.

On initialization, $\mathbf{M}[-1|-1]$ is set to the covariance $\mathbf{C}_s$ of the random initial state $\boldsymbol{s}[-1]$. Particularly in our implementation, due to the lack of knowledge about the initial state, we populate $\mathbf{C}_s$ with arbitrarily large values, such that the filter starts with little confidence in the state. Moreover, we initialize the state vector $\boldsymbol{s}[-1]$ with the first available time and frequency offset measurements, from (7) and (14), respectively, rather than assigning a random realization of a random vector with PDF $\sim \mathcal{N}(\mu_s, \mathbf{C}_s)$. This heuristic choice is meant to achieve faster convergence, and it is also motivated by our unawareness of the actual μ_s and $\mathbf{C}_s$.

We adopt another heuristic approach to define the state noise covariance matrix $\mathbf{Q}$. As discussed in [39], this matrix can be determined based on the offline characterization of the free-running local oscillator, for example, through Allan variance curves. However, this is a non-scalable approach, as it requires a manual procedure for each device. An alternative is to tune $\mathbf{Q}$ with some degree of trial and error. As discussed in Section V, our implementation in this work tries a range

of matrices $\mathbf{Q}$ and picks the one that results in the best performance. Nevertheless, one should note that it is crucial to capture the statistics of the actual state noise for better performance. Our inability to do so in a scalable and practical way can limit the achievable performance.

Meanwhile, the measurement noise variance σ_w^2 depends mostly on the PDV experienced by PTP. Based on (7), and assuming that the m-to-s and s-to-m delays are independent, wide-sense stationary, and white discrete-time random processes, it follows that:

$$\sigma_w^2 = \frac{\text{Var}\{d_{ms}[n]\} + \text{Var}\{d_{sm}[n]\}}{4}, \tag{45}$$

where Var {} denotes the variance. Advantageously, σ_w^2 coincides with the variance of the two-way delay measurement from (8), which the PTP slave node can compute in practice.

Moreover, note that the measurement noise variance in (45) depends solely on the delay variances and not their averages. Thus, the delay fluctuations due to queuing and (to a less extent) processing delays determine it, whereas the static delay asymmetries (see Table 1) do not influence it. Nevertheless, static asymmetries do harm the KF assumption of zero-mean measurement noise. Secondly, note that, in contrast to [39], we assume that the PDV determines the measurement noise in (45) while neglecting other effects such as the timestamping granularity.

Ultimately, the given KF formulation is limited by three main factors. The first refers to the state noise model, which considers only two forms of phase noise, whereas other phase noise sources may be significant. The second comes from the non-Gaussian and non-zero-mean measurement noise, which limits the estimates to sub-optimality. The third refers to the difficulty of defining the state noise covariance matrix $\mathbf{Q}$ and initial state $\mathbf{s}[-1]$ accurately.

D. TIME OFFSET DRIFT ESTIMATION

Section IV-A discusses the approach of (22), which benefits from time offset drift estimates. In this work, we estimate the time offset drifts as follows:

$$\hat{\Delta}_x[n] = \hat{y}[n]\,(t_1[n] - t_1[n-1])\,, \tag{46}$$

where $\hat{y}[n]$ represents the frequency offset estimate. Furthermore, we obtain $\hat{y}[n]$ based on the one-way formulation in (14), instead of (12). This is not a general guideline, but a choice that leads to better performance in our experiments, where the m-to-s PTP messages experience less PDV.

To further alleviate the PDV noise, we process $t_{21}[n]$ using an extra layer of window-based filtering. As illustrated in Fig. 2, this processing relies on two observation windows spaced by an interval of N samples, each containing K values of $t_{21}[n]$. The front window operates on the most recent K samples, whereas the back window operates on samples in the past. Ultimately, the estimate is obtained by:

$$\hat{y}[n] = \frac{\underset{0\le j<K}{\xi}\{t_{21}[n-j]\} - \underset{0\le j<K}{\xi}\{t_{21}[n-N-j]\}}{t_1[n] - t_1[n-N]}. \tag{47}$$

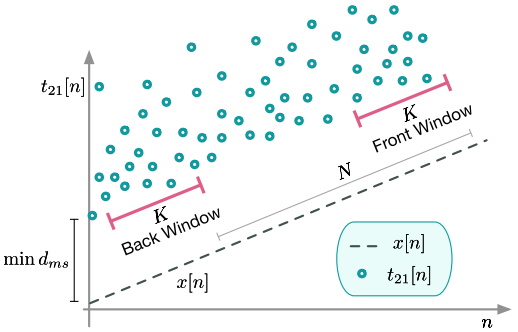


FIGURE 2. Window-based filtering strategy used for frequency offset estimation.

where ξ is an arbitrary operator. In this work, we explore specifically the minimum and maximum operators in (47).

The same tradeoffs discussed in Section IV-A apply to (47). While the window length K must be short enough to capture a constant time offset, it is also helpful to have large windows to increase the chances of finding, e.g., minimally or maximally-delayed packets in each window. Nevertheless, in contrast to (23), where drift correction allows larger windows, (47) does not have any solution for drifts. After all, (47) is a pre-requisite for estimating the drifts in (46). Hence, we explore relatively short windows (small K) in (47).

V. PROCESSING ARCHITECTURE

A major goal in this work is to compare the synchronization algorithms fairly by applying them simultaneously on the same data. To do so, we developed the open-source Python-based PTP dataset analysis library (PTP-DAL), which implements all processing stages described thus far, with the architecture illustrated in Fig. 3. Its analysis tool first reads a dataset containing timestamps and labels collected by our FPGA-based testbed. Then, it computes the corresponding time offset measurements according to (6) and the timestamp difference metrics from (5), which are fed into the algorithms from Section IV. In the

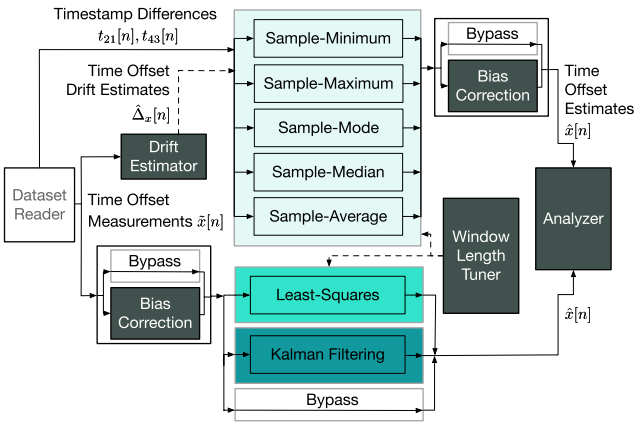


FIGURE 3. Software architecture implemented on PTP-DAL to process the timestamps and labels of a dataset acquired from the testbed.

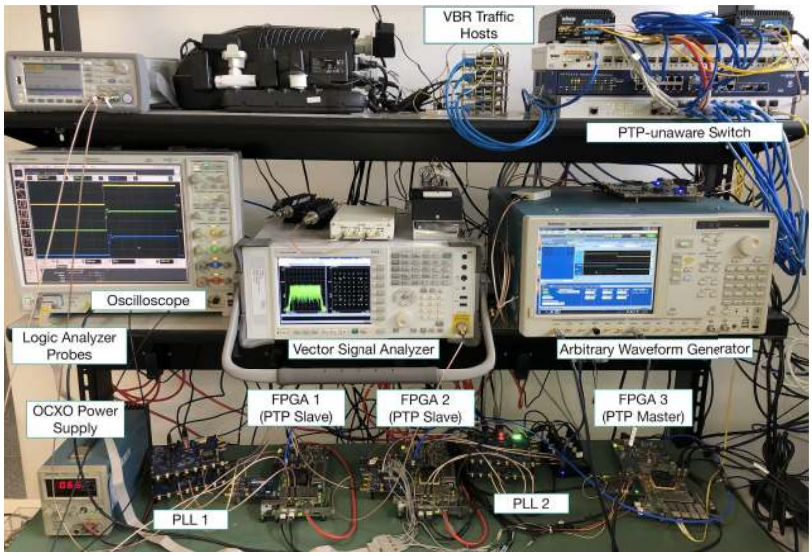


FIGURE 4. Picture of the FPGA-based synchronization testbed.

end, the results of each algorithm are compared within the Analyzer module.

Fig. 3 shows the stack of time offset estimation algorithms implemented on PTP-DAL. Note that the algorithms from Section IV-A rely on time offset drift compensation, using equations (22) and (23). Thus, they receive the estimates from the drift estimator module discussed in Section IV-D. Meanwhile, the LS and KF algorithms process the time offset measurements $\tilde{x}[n]$ directly.

The architecture in Fig. 3 includes optional bias correction stages, which are explored in Section VII. These stages shift time offset measurements or estimates by fixed values computed from the dataset labels. For the algorithms from Section IV-A, which compute the time offset estimates based on timestamp differences $t_{21}[n]$ and $t_{43}[n]$, the bias correction module shifts the time offset estimates produced by (23), i.e., the algorithms' outputs. In contrast, for the algorithms that process the time offset measurements $\tilde{x}[n]$ directly (LS and KF), the bias correction stage shifts the algorithm's input.

Furthermore, the bias correction module applies a distinct correction to each method from Section IV-A, depending on the operator $\xi\{\}$ in (23). The rationale is that $\xi\{\}$ governs the asymmetry resulting from approximation (19). Hence, the bias correction module computes the asymmetry as follows:

$$\hat{b} = \frac{\xi\{d_{ms}[n]\} - \xi\{d_{sm}[n]\}}{2}. \tag{48}$$

For example, this expression returns the asymmetry between the minimum m-to-s and s-to-m PTP delays when using the sample-minimum approach. The remaining algorithms (LS and KF) experience biases corresponding to the average noise $w[n]$ from (7), which also affects the sample-average strategy. Hence, the bias correction module shifts them by the average delay asymmetry.

PTP-DAL also includes the window length tuner module, which finds the best observation window length N for the algorithms from Section IV-A and LS. This module runs each window-based algorithm for a range of window lengths and returns the length that yields the best performance in terms of the maximum absolute time error (max|TE|) [51]. In the end, all window-based algorithms are compared using their (distinct) best window lengths.

Other modules also have their optimizers, such as the KF and drift estimator. The Kalman filter's optimizer sweeps a range of state noise covariance matrices $\mathbf{Q}$ and chooses the one that yields the lowest max|TE|. Similarly, the drift estimator's optimizer sweeps a range of N and K in (47) until the resulting drift estimates from (46) present minimal error relative to the true drifts observed on dataset labels. Ultimately, PTP-DAL allows the comparison of algorithms based on their best (reasonably optimized) configurations.

VI. TESTBED

Our testbed is composed of three Xilinx Virtex-7 FPGAs. One of them is the PTP master clock, and the others are the PTP slave clocks. Fig 4 shows the FPGAs, the switches, and several measurement instruments that compose the testbed, some of which are out of scope. The interested reader can refer to the description of an early version of this testbed in [52] and a more recent usage regarding radio-frequency timing alignment experiments in [53].

In this section, we present the FPGA design choices for the acquisition of timestamp datasets. Next, we describe the testbed network and the sources of BG traffic used on experiments. Finally, we discuss the PTP delays that are expected based on the adopted testbed configurations.

A. FPGA DESIGN

The FPGA design of the PTP slave clock supports two distinct oscillator options to drive the RTCs: an onboard

crystal oscillator (XO), available on the adopted Xilinx VC707 board, and an external oven-controlled crystal oscillator (OCXO). As shown in Fig. 4, each slave clock has an associated Analog Devices AD9548 phase-locked loop (PLL) board, which, among other tasks (see [52]), supplies a clock signal synthesized based on an OCXO.

The feature of a configurable oscillator allows the evaluation of two classes of RTC stability and accuracy. The XO presents a frequency stability specification of ±50 ppm, whereas the OCXO stays within ±5 ppb. In the experimental section, we analyze results using the two oscillator options.

In contrast, our master clock only takes one driving clock for its RTC. This reference comes from a high-stability 125 MHz differential clock signal synthesized by an arbitrary waveform generator (AWG), shown in Fig 4. This is meant to be the primary reference clock (PRC) of the network [45].

Furthermore, each PTP slave clock in the testbed includes two RTCs in the FPGA fabric. The first is the conventional PTP-synchronized RTC, which is the source of time for the timestamping of PTP messages. For brevity, we refer to this instance as the *PTP RTC*. The second is a self-developed RTC that synchronizes to a pulse per second (PPS) reference signal, referred to as the *PPS RTC*. This dual-RTC architecture is illustrated in Fig. 5, and it is typical on devices featuring primary and backup timing sources, such as in the case of APTS slave clocks (see Appendix A in [17]).

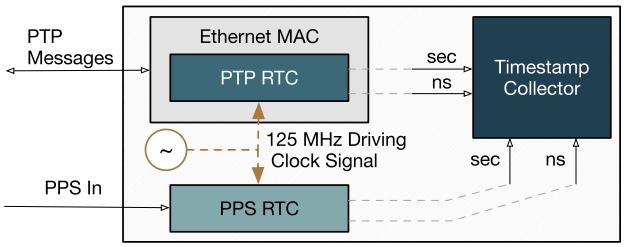


FIGURE 5. Dual-RTC architecture used on the slave clocks.

As illustrated in Fig. 6, our master clock outputs a PPS signal to the PTP slaves, which then synchronize their PPS RTCs to this signal. Ultimately, the slaves synchronize their PTP and PPS RTCs through entirely independent mechanisms, the former through PTP over the network, and the latter through the reliable PPS signal. Nevertheless, both the PTP and the PPS timing references come from the master clock's time base. Thus, if the synchronization mechanisms were ideal, the two RTCs would align perfectly.

A fundamental aspect of the FPGA design is that the PTP and PPS RTCs are driven by the same clock signal of $f_{\text{nom}} = 125$ MHz, as shown in Fig. 5. Ultimately, this choice allows an accurate correspondence between timestamps taken from the two RTCs. Also, with this frequency of 125 MHz, by default, the two RTCs increment in steps of $1/f_{\text{nom}} = 8$ ns. Correspondingly, the inaccuracy of each PTP timestamp due to the time quantization ranges from 0 to 8 ns.

To reduce the timestamp granularity, the PPS RTC developed in-house uses a double flip-flop re-synchronizer circuit

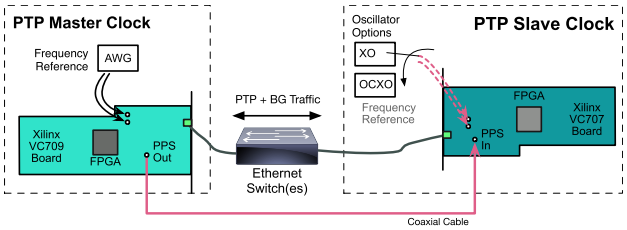


FIGURE 6. PPS connection and the frequency references supplied to the FPGAs, with XO and OCXO options on the slave clocks.

similar to the approach in [54]. This circuit observes both rising and falling edges of the RTC's driving clock, and so it halves the timestamp granularity. In the end, the PPS RTC features a timestamping accuracy nominally within ±2 ns.

When receiving the PPS signal from the master, the slaves continuously measure the time error (TE) between their local PPS RTCs and the rising edges of the input signal. With the measured TE, our FPGA's firmware estimates the local frequency offset relative to the master and compensates it by adjusting the increment step of the PPS RTC. This process happens through a software implementation of a proportional-integral loop. Once the loop locks, we typically (99.8% of the time) observe a steady-state synchronization error within ±4 ns with the OCXO and ±8 ns with the XO.

In addition to the timestamp granularity, a significant timestamping uncertainty source comes from variable latency processing stages before the timestamping point. This is not a concern for PPS synchronization because the timestamps are taken directly based on the input PPS signal. In contrast, it is a challenge for PTP because the timestamping usually happens inside the medium access control layer (MAC) and is adjusted back to the corresponding time at the physical medium interface [55]. Thus, the timestamps are subject to variable latency stages from the PHY [23], [56].

Our clocks implement the Ethernet MAC within the FPGA fabric and rely on an external PHY chipset (Marvell 88E1111 on the slaves). They timestamp the PTP messages within the MAC when the start of frame delimiter of the PTP frame (carrying the message) is observed on a plane that lies after MAC stages with fixed and known latency. Subsequently, each timestamp is adjusted in software to the corresponding time at the gigabit media-independent interface (GMII) interface. Thus, the timestamps are nanosecond-accurate and represent the GMII time. Nevertheless, the timestamps do not represent the time at the physical medium. Correspondingly, any asymmetry or variable latency incurred within the master's or slave's PHY Tx and Rx paths contribute to the total path delay. The characterization of such PHY effects is beyond the scope of this work.

B. LABELED DATA ACQUISITION

A key feature of the testbed is that it can capture the actual one-way delay of each PTP message and the true slave time offset during each message exchange. This is possible because the slave clocks can take simultaneous snapshots

of their PTP and PPS RTCs, and because the PPS RTC is accurately synchronized to the master time.

For each two-way PTP exchange, the slave collects the standard set of four timestamps taken by the PTP RTCs and two additional timestamps from the slave's PPS RTC. As shown in Fig. 1, the two timestamps that are normally taken on the slave side are $t_2[n]$ and $t_3[n]$. Correspondingly, our slave takes timestamps $t'_2[n]$ and $t'_3[n]$ from the PPS RTC at the exact clock cycle when it takes $t_2[n]$ and $t_3[n]$ from the PTP RTC. In the end, the n-th entry of a dataset acquired from the testbed contains the information in Table 3.

TABLE 3. Example dataset entry corresponding to a PTP message exchange.

n	$t_1[n]$	$t_2[n]$	$t_3[n]$	$t_4[n]$	$t'_2[n]$	$t'_3[n]$

With the additional timestamps from the PPS RTC, $t'_2[n]$ and $t'_3[n]$, the truth labels can be computed as follows:

$$\acute{x}[n] = t_2[n] - t'_2[n] \tag{49}$$

$$\acute{d}_{ms}[n] = t'_2[n] - t_1[n] \tag{50}$$

$$\acute{d}_{sm}[n] = t_4[n] - t'_3[n], \tag{51}$$

where $\acute{x}[n]$, $\acute{d}_{ms}[n]$, and $\acute{d}_{sm}[n]$ represent the time offset, m-to-s delay, and s-to-m delay truth labels, respectively. The rationale is that the PPS RTC accurately represents the master time. Thus, for example, $t_2[n]-t'_2[n]$ represents the difference between the slave time $t_2[n]$ upon the arrival of the n-th PTP message (according to the PTP RTC) and the master time $t'_2[n]$ (according to the PPS RTC) when the PTP arrival timestamp is taken. The error associated with such labels is the PPS synchronization error, which, as mentioned in Section VI-A, ranges from ± 4 ns (OCXO) to ± 8 ns (XO).

A noteworthy aspect is that there is no uncertainty on the matching between a timestamp from the PTP RTC (e.g., $t_2[n]$) and a timestamp from the PPS RTC (e.g., $t'_2[n]$). Our hardware drives the two RTCs using the same clock signal, as illustrated in Fig. 5. Hence, the two RTCs reside in the same clock domain, and there is no need for clock domain crossing between them. This design choice avoids timing uncertainties and, ultimately, allows the sampling of the two RTCs precisely at the same clock cycle.

C. TESTBED NETWORK AND BG TRAFFIC SOURCES

As illustrated in Fig. 7, our testbed connects the FPGA devices through a PTP-unaware GbE network with a configurable number of hops. Each network hop is implemented on an independent port-based virtual local area network (VLAN) on the PTP-unaware switch (shown in Fig. 4). We explore from one to four hops in this work, as this range represents well-controlled PTS networks with few hops [16]. Furthermore, we adopt a tree topology, with the two slave clocks connected to the same hop. For example, Fig. 7 illustrates the two slaves connected to the fourth hop.

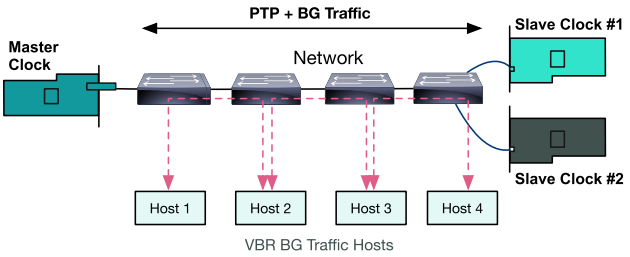


FIGURE 7. Testbed's PTP-unaware GbE network with hosts used for the generation of BG traffic.

The given network carries only two types of traffic: PTP and BG traffic. The latter, in turn, can be generated in two forms: constant bit rate (CBR) and variable bit rate (VBR). The CBR BG traffic is generated directly by the FPGA devices in hardware with precise configurations in terms of bit rate and packet inter-departure intervals. In contrast, the VBR BG traffic is generated with significantly lower precision using the *iperf* application on auxiliary Raspberry Pi Linux hosts, highlighted as the *VBR traffic hosts* in Fig. 4.

The CBR BG traffic generated by the FPGA devices consists of fixed-length packets with periodic packet departures. As illustrated in Fig. 8, the master sends two BG packets on every serving period, one to each slave. In contrast, the slaves send a single packet to the master every serving period.

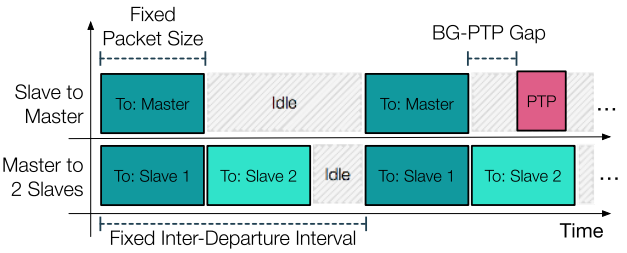


FIGURE 8. CBR BG traffic pattern in the slave-to-master and master-to-slave directions.

Note that the CBR BG packets follow the same path as the PTP packets (except for the last hop, discussed later). This configuration is referred to as *in-line* BG traffic in [20]. In contrast, the VBR BG traffic is generated and consumed by devices other than the FPGAs. Hence, each VBR stream runs over an isolated portion of the path between the PTP clocks.

As shown in Fig. 7, each *VBR BG traffic host* injects one-way UDP-based VBR BG traffic on the switch and receives the stream from the previous VBR traffic host. For example, Host 2 receives the stream from Host 1 and injects another independent stream towards Host 3. This pattern is referred to as *cross-traffic* in [20], [21]. Following the literature, we configure this traffic according to a model from ITU-T Recommendation G.8261 [57]. Nevertheless, unlike [20], [21], we experiment with cross-traffic in both the m-to-s and s-to-m directions, and we adopt the data-centric Network Traffic Model 2 instead of the voice-centric Model 1.

D. EXPECTED PTP DELAYS

Next, we define the expectations in terms of PTP delays, given that they are determinant to the synchronization performance. This is more easily attainable for the CBR BG traffic due to its accurate hardware-based generation and tractable pattern with periodic transmissions. Hence, we take the CBR source as an example of BG traffic within a well-controlled PTS network with predictable delays, which we discuss next.

A key figure for the analysis of delays is the gap between the start of the PTP message and the end of the preceding BG frame, denominated as the *BG-PTP* gap in Fig. 8. As discussed in [35], assuming that the PTP messages are shorter than the BG messages, the BG-PTP gap reduces after every store-and-forward switching stage.[3] More specifically, it reduces by $(\tau_{bg} - \tau_{ptp})$ on each hop, where τ_{bg} and τ_{ptp} are the BG and PTP packet transmission (or serialization) delays, respectively. Eventually, after sufficient hops, the PTP message approaches the BG packet completely. From this point on, the PTP message has to wait for the full serialization of the preceding BG packet on every subsequent store-and-forward stage. Conversely, while the BG-PTP gap is still non-zero, the PTP message does not experience queuing delays and can be *forwarded* immediately after being fully *stored*.

The starting BG-PTP gap determines how soon the PTP message experiences queuing delays. The worst-case is when the PTP message departs with the so-called zero-gap (ZG) condition, i.e., already immediately after a BG packet. This scenario leads to the following one-way delay model:

$$d_{ms,zg}[n] = \eta(\tau_{bg}) + \sum_{j=0}^{\eta-1} (\gamma[j,n] + \beta_{ms}[j]) \,, \qquad (52)$$

where η is the number of network hops, $\gamma[j,n]$ is the n-th random processing delay within the j-th hop, and $\beta_{ms}[j]$ models the static hardware latency associated with the j-th hop in the m-to-s direction. For instance, the latter includes the Ethernet PHY latency on ingress and egress ports. For simplicity, this model ignores the propagation delays.

In the model of (52), term $\eta(\tau_{bg})$ includes both the transmission and queuing delays. The PTP message experiences transmission delay τ_{ptp} on every hop. By the time the PTP message is fully *stored* (after τ_{ptp}), a wait of $(\tau_{bg} - \tau_{ptp})$ remains until the outbound link finishes the serialization of the preceding BG packet. Under the ZG condition, the PTP message experiences a queuing delay of $(\tau_{bg} - \tau_{ptp})$ on each hop. Thus, in total, the two delays compound to $\eta(\tau_{bg})$.

Moreover, the model of (52) holds in the m-to-s direction when PTP shares the network solely with in-line CBR BG traffic in the specific tree topology of Fig. 7, and provided that condition $\tau_{bg} > \tau_{ptp}$ holds. This is because, in this scenario, the m-to-s PTP messages do not experience contention. In contrast, the model does not hold in the s-to-m direction, where the PTP messages can experience contention.

[3]In this work, we rely and focus on store-and-forward switching, as opposed to cut-through switching.

The contention of s-to-m PTP packets occurs on the switching node where the two slave clocks are connected (e.g., the fourth node in Fig. 7), referred to as the *aggregation node*. The packets from the two slaves compete for the shared outbound link towards the master. For example, if a PTP packet from Slave 1 arrives in the aggregation node slightly after a BG packet from Slave 2, the PTP packet can experience an extra queuing of τ_{bg}. Thus, under the ZG condition, the worst-case s-to-m delay becomes:

$$d_{sm,zg}[n] = (\eta + 1)(\tau_{bg}) + \sum_{j=0}^{\eta-1} (\gamma[j,n] + \beta_{sm}[j]) \,, \qquad (53)$$

where $\beta_{sm}[j]$ models the static hardware latency associated with the j-th hop in the s-to-m direction.

Equations (52) and (53) define the worst-case (highest) expected PTP delays. However, the PTP messages can experience significantly lower delays when departing with the maximum possible BG-PTP gap and in the absence of contention. Thus, due to the random placement of PTP packets relative to BG frames, we expect a PDV corresponding to the BG-PTP gap variations. Also, in both directions, we expect additional fluctuations up to the order of τ_{bg}.

In the s-to-m direction, queuing delay variations in the order of τ_{bg} are expected due to the random contention in the aggregation node. In contrast, in the m-to-s direction, queuing delay variations in the order of $(\tau_{bg} - \tau_{ptp})$ happen because the CBR BG packets do not necessarily follow the PTP path in the last hop. For example, a PTP message going towards slave clock 1 (see Fig. 7) can depart after a unicast-addressed BG frame sent to slave clock 2. In this case, under the ZG condition, the PTP message waits for the preceding BG frame's serialization in each hop, except the last one. Once the PTP message is fully *stored* in the last switching stage, it can be *forwarded* immediately (with no queuing delay), while the BG frame is forwarded through another port (in this example to slave 2). This event happens randomly due to the random placement of PTP messages on the wire.

VII. RESULTS

In this section, we compare the synchronization algorithms discussed in Section IV under various scenarios. First, we present the results achieved when PTP is the only traffic in the testbed network, namely without BG traffic. Subsequently, we show results under in-line CBR and cross-traffic VBR BG traffic. In all cases, we consider hour-long datasets acquired with the two PTP slave clocks running 128 delay request-response exchanges per second with the XO and OCXO options. Moreover, we assess the performance in terms of the max|TE| observed over every minute.

The choice of 128 PTP exchanges per second represents the maximum packet rate in many PTP profiles, such as the PTS profile [13]. With this choice, the goal is to evaluate the algorithms under the most favorable conditions for performance. Meanwhile, the communications load that results from using

the highest PTP packet exchange rate is still negligible in many practical use cases.

Furthermore, our implementation on PTP-DAL computes all window-based algorithms using overlapping windows, unlike in the tractable formulation adopted thus far. Hence, in the experiments that follow, the algorithms output one estimate for every new PTP exchange, except for the initial $N-1$ exchanges. Furthermore, the experiments explore observation window lengths ranging from $N = 4$ to 65536. With 128 PTP exchanges per second, this range corresponds to durations from 31.25 milliseconds up to 512 seconds.

Moreover, although each dataset consists of an hour-long acquisition, we choose to skip the initial 25% of the results in each experiment to disregard transients. This choice corresponds to 15 min, namely a duration longer than all observation windows considered in the experiments. Consequently, the results that follow show only 45 min.

Lastly, as discussed in Section VI-C, the experiments that follow explore from one to four network hops. Table 4 summarizes the experimental parameters and scenarios.

TABLE 4. Parameters and configurations adopted in the experiments.

Parameter	Value
PTP exchange interval	$T = 7.8125$ ms
Window lengths	From $N = 4$ to 65536
Network hops	From $\eta = 1$ to 4
Oscillators	XO and OCXO
BG traffic scenarios	A) No BG traffic B) In-line CBR BG traffic C) VBR BG cross-traffic

A. PTP WITHOUT BG TRAFFIC

To start, Fig. 9 shows the distributions of the true one-way PTP packet delays observed in the absence of BG traffic for all hop configurations (from one to four) and with both oscillator options. These delays are computed using (50) and (51), namely based on the dataset labels. Note that the delay distributions present a similar shape and support. More specifically, all scenarios have a standard deviation (SD) of approximately 160 ns and span roughly between 590 to 700 ns. Furthermore, note that each hop adds nearly 3.7 to 4.3 μs of delay. This amount includes the processing delay, the constant PTP transmission delay τ_{ptp} (640 ns in this case), and the hardware latency associated with the hop. Besides, note that, in the absence of BG traffic, the processing delay becomes the predominant variable delay component and determines the shape of the distributions.

Although the distributions in Fig. 9 present similar shapes, the difference between the average m-to-s and s-to-m delays varies in each hop configuration. This difference refers to the static delay asymmetry component modeled in (10), which varies per hop because each port of the PTP-unaware switch introduces different static hardware latencies (e.g., Ethernet PHY latencies). For example, note that the asymmetries observed with the XO and OCXO match closely on

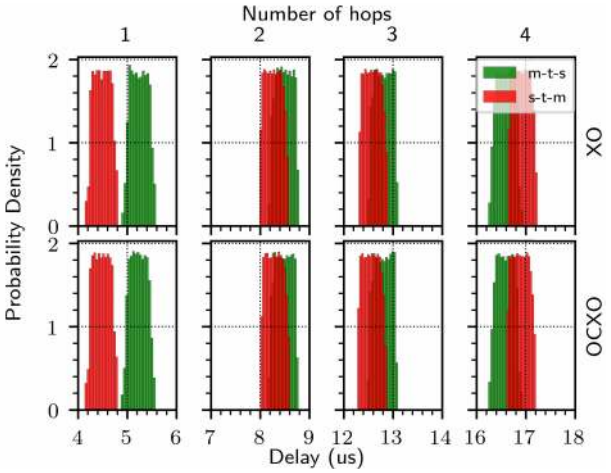


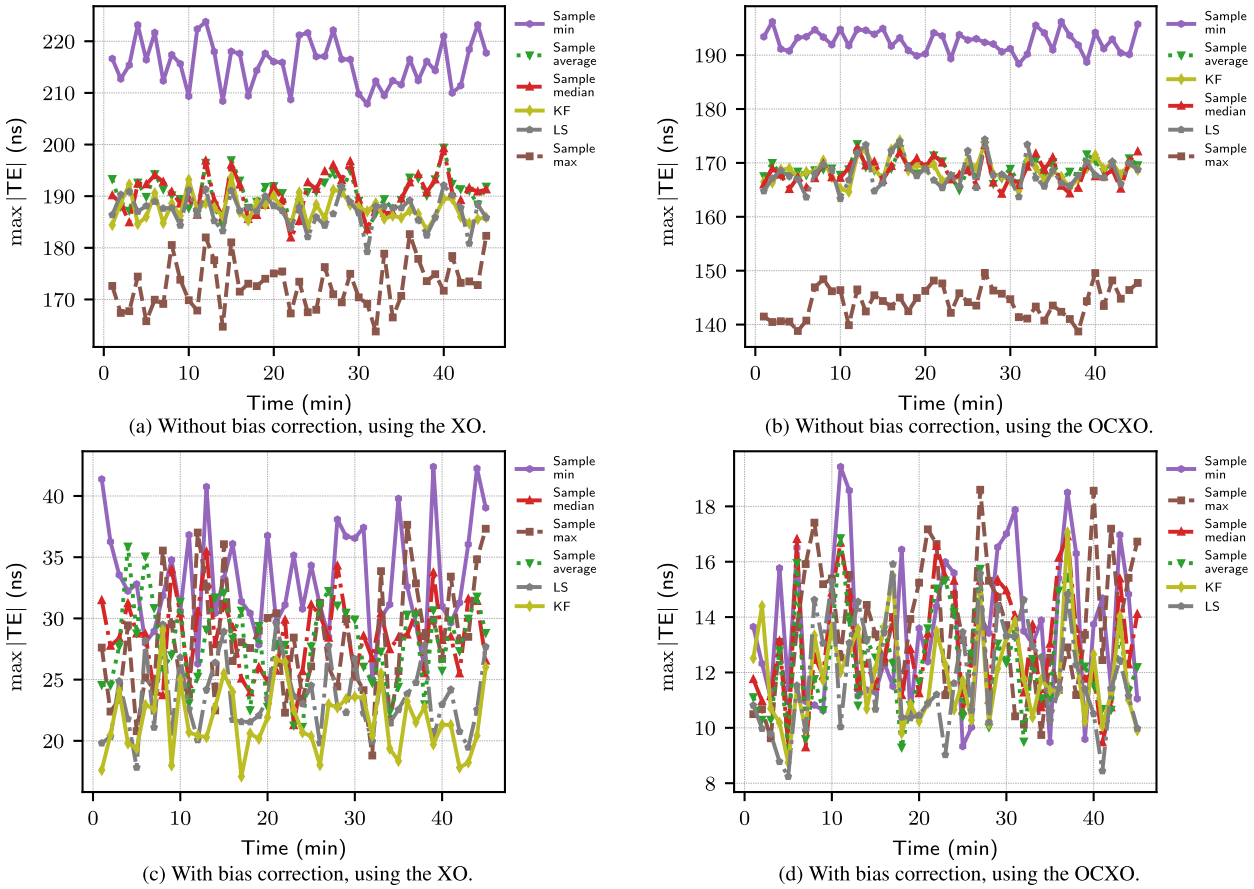
FIGURE 9. PTP one-way delay distributions in the absence of BG traffic.

each hop configuration (using the same ports). In contrast, when comparing datasets with a different number of hops (different ports), the asymmetries diverge significantly.

In a practical PTS deployment, such varying asymmetry contributions are expected. Ordinarily, switches (and especially PTP-unaware ones) provide no guarantees regarding the asymmetry between the transmit and receive paths of each of its ports. Thus, the PTP slave has to live with the asymmetry. Alternatively, in an APTS deployment, the delay asymmetry experienced by the PTP messages can be measured while the slave clock is locked to a GNSS reference. Then, upon the GNSS signal's loss, when the APTS slave clock switches to the PTP (backup) timing reference, it can use the previously measured PTP delay asymmetry to adjust the PTP timestamps and the corresponding measurements.

Next, we analyze the synchronization performance achieved over four hops. Fig. 10 shows the max|TE| performance achieved by the processing algorithms discussed in Section IV, with max|TE| measured over every minute of the experiment. It contrasts the case where the slave does not have any means to correct the bias to the case where the slave can calibrate and correct the bias, as in the APTS scenario. In this comparison, we omit the sample-mode to better visualize the curves that achieve similar max|TE| values. The sample-mode performance is significantly worse and unstable because the delay distributions in Fig. 9 do not present a prominent mode (the distributions are flat over nearly 400 ns). Similarly, we omit the max|TE| curve referring to the raw measurements from (6), which lies around 470 ns without bias correction and 300 ns with bias correction.

In Fig. 10a and 10b, which show the performance without bias correction, note that the sample-maximum approach using (23) achieves the best performance. Unlike in the evaluation of [19], this result is not due to a high network load (there is no BG traffic in this case). Instead, it is because, in this experiment, the asymmetry between the maximum m-to-s and s-to-m delays is lower than the other asymmetries,



(a) Without bias correction, using the XO.
(b) Without bias correction, using the OCXO.
(c) With bias correction, using the XO.
(d) With bias correction, using the OCXO.

FIGURE 10. max|TE| results obtained in the experiment without BG traffic over four network hops.

as summarized in Table 5, and because the biases due to the delay asymmetries are more significant than the estimation variance presented by all algorithms. Hence, the estimator experiencing the lowest asymmetry performs best.

In fact, the estimations in Fig. 10 present very low variance, as is expected in the absence of BG traffic. Thus, in Fig. 10a and Fig. 10b, the sample-average, sample-min, sample-max, and sample-median curves converge to values near their corresponding asymmetries in Table 5. The LS and KF results also converge to values near their expected asymmetries, namely to the average asymmetry (also in Table 5). This convergence is tighter in Fig. 10b, with the OCXO. In contrast, with the XO (Fig. 10a), the max|TE| curves are slightly worse (by around 25 ns) because the drift estimates used in (23) and the estimates produced by LS and KF become less accurate when the LO is less stable.

Next, Fig. 10c and 10d present the performances when applying bias correction. In this case, all algorithms achieve similar max|TE| values, with minor differences. The LS and KF estimators were slightly superior in both cases. Furthermore, as in the previous scenario, the performance achieved with the OCXO (Fig. 10d) is superior to the one achieved with the XO (Fig. 10c). More generally, both configurations with bias correction achieve excellent synchronization accuracy,

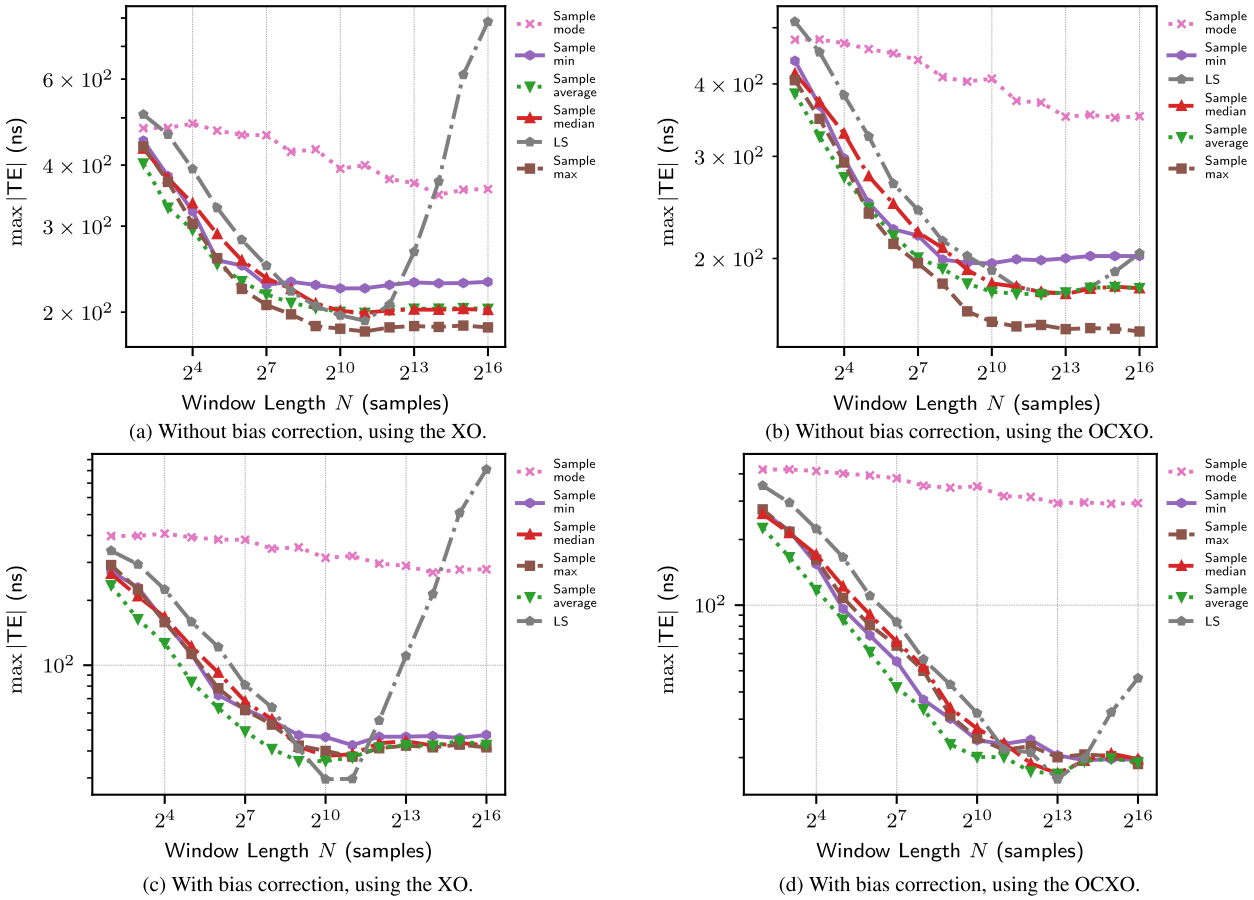
TABLE 5. Static delay asymmetries over four hops without BG traffic.

Operator	Average	Min	Max	Median
Asymmetry (ns)	−168	−197	−145	−167

despite the network's lack of PTP support. These results illustrate the achievable performance when the PTP-unaware network can be dedicated exclusively to PTP traffic.

The results in Fig. 10 are based on optimized parameters set for each algorithm. Fig. 11 illustrates specifically the optimization of the observation window length N used by the algorithms from Section IV-A and LS, including the sample-mode approach, which was omitted in Fig. 10. More precisely, Fig. 11 shows the global max|TE| achieved by each algorithm for varying power-of-two window lengths ranging from $N = 4$ to $N = 65536$.

In the absence of strong PDV (i.e., without BG traffic), the max|TE| performance as a function of the window length often presents a convex shape. That is, it is helpful to increase the observation window up to a certain point. After that, longer windows do not bring performance improvements and can even degrade performance. This convex shape is more visible for LS, given that, as discussed in Section IV-B, the



(a) Without bias correction, using the XO.

(b) Without bias correction, using the OCXO.

(c) With bias correction, using the XO.

(d) With bias correction, using the OCXO.

FIGURE 11. max|TE| performance according to the observation window length on the scenario without BG traffic and with four network hops.

LS estimator requires observation windows where the frequency offset remains constant. Once the observation window becomes too large, this requirement starts to fail, and the resulting performance degrades. Furthermore, note that this turning point occurs for a shorter window length when using the XO, due to the worse frequency stability.

An essential benefit of the drift compensation scheme in (23) is that it enables long observation windows. Fig. 11 shows that the algorithms using this equation (sample-min, max, average, median, and mode) tolerate longer windows than LS. After a certain length covering enough of the delay statistics, they tend to reach a plateau, but the performance does not degrade significantly as the window grows further. This property holds as long as the drift estimates from (46) do not accumulate significant error over the windows.

B. PTP UNDER CBR BG TRAFFIC

Next, we evaluate the synchronization performance under the in-line CBR BG traffic described in Section VI-C. More specifically, we configure the CBR BG traffic with a packet inter-departure interval of 4.16 μs and each BG packet with 236 bytes at layer-1 (including all the Ethernet framing bytes). Thus, each BG frame has a transmission delay τ_{bg} of 1.888 μs over GbE. When serving two slaves, the two BG frames sent per period occupy 3.97 μs of the inter-departure interval (including 96 ns inter-packet gaps). Hence, the idle interval illustrated in Fig. 8 is roughly 0.2 μs.

Moreover, with two BG packets every 4.16 μs (one to or from each slave), the total network utilization is around 95%. Hence, the evaluation that follows illustrates the synchronization performance under very high BG utilization, with background traffic composed of relatively short packets. The CBR BG traffic parameters are summarized in Table 6.

TABLE 6. In-line CBR BG traffic parameters.

Parameter	Value
Packet transmission delay	$\tau_{bg} = 1.888$ µs
Inter-departure interval	4.16 µs
Network utilization	95%

1) PTP DELAY ANALYSIS

In this scenario, we can anticipate the worst-case PTP delays using (52) and (53). For simplicity, we can assume a constant processing delay per hop of $\gamma[j,n] = 3.3$ μs and a constant hardware latency per hop of $\beta_{ms}[j] = \beta_{sm}[j] = 300$ ns (see, e.g., [23]). Using these assumptions, the ZG m-to-s and s-to-m become as summarized in Table 7.

TABLE 7. Expected worst-case (zero-gap) m-to-s and s-to-m one-way PTP delays under in-line CBR BG traffic from one to four hops.

Hops	1	2	3	4
$d_{ms,zg}[n]$ (µs)	5.49	10.98	16.46	21.95
$d_{sm,zg}[n]$ (µs)	7.38	12.86	18.35	23.84

Fig. 12 shows the actual delay distributions observed within testbed acquisitions with in-line CBR BG traffic. Compared to Fig. 9, the PDV is substantially higher in Fig. 12. Among all acquisitions, the average SD and support in the m-to-s direction are 518 ns and 1961 ns, respectively. As discussed in Section VI-D, the delay variations in this direction include the BG-PTP gap fluctuations (within the idle interval of 0.2 μ s) and the last-hop delay fluctuation of 1.25 μs (i.e., $\tau_{bg} - \tau_{ptp}$). Furthermore, the PDV includes the inherent processing delay variations, which, from Section VII-A, are expected to span up to 700 ns. These three fluctuation components totalize to 2.15 μs. Hence, the observed m-to-s delay variation is within the expected range. Meanwhile, the s-to-m delays present SDs ranging from 495 ns to 738 ns and average support of 2.78 μs. These figures also closely follow the expectations from Section VI-D.

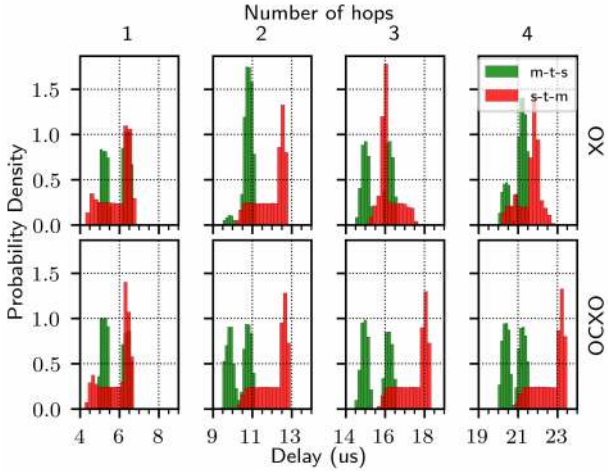


FIGURE 12. PTP one-way delay distributions under in-line CBR BG traffic.

Nevertheless, it must be emphasized that the distributions in Fig. 12 are not accurately repeatable. Instead, they are illustrative realizations. The distributions vary per experiment due to the asynchronous timing relationship between the PTP and BG packet departures, manifested in two forms. The first is intra-FPGA and refers to the interval between PTP packets and the preceding BG packets generated by the same FPGA, namely the *BG-PTP* gap discussed in Section VI-D. The second asynchronous relationship is inter-FPGA and refers to the relative departure time difference between the PTP packets of one FPGA and the BG packets of the other.

Both PTP and BG streams consist of periodic packets accurately generated by the FPGAs. In the given experiment, with the BG packet departure periodicity of 4.16 μs, the slave clocks send and receive nominally 1875 BG packets in the course of a PTP exchange interval (of 7.8125 ms). However, the two streams operate in different clock domains that can drift over time relative to each other, such that the BG-PTP timing relationship changes over time. Moreover, the streams are not enabled simultaneously on the same device and neither among the group of FPGAs, which yields a random initial timing relationship between them.

For example, in the s-to-m direction, the initial timing relationship between the PTP packets of one slave clock and the BG packets of the other slave determines the initial amount of contention-induced queuing delays experienced by PTP packets on the aggregation node (see Section VI-D). When the clock domains governing the packet generation logic do not drift significantly over time (typically the case, especially with the OCXO), this initial timing relationship tends to last throughout the experiment. As a result, we observe a more frequent amount of contention determining the peak of the s-to-m delay distributions, but which varies among experiments. Meanwhile, in the m-to-s direction, the instant when BG and PTP packets turn on influences the likelihood of PTP packets departing after a unicast BG packet addressed to slave clock 1 or slave 2. As such, it determines whether the PTP packet experiences an extra queuing delay in the last hop, as discussed in Section VI-D. With all of these effects, the distributions in Fig. 12 are not repeatable.

On the other hand, we can observe repeatable patterns in Fig. 12. Firstly, the m-to-s delays are typically bimodal due to the last-hop delay fluctuation explained in Section VI-D. Secondly, in the s-to-m direction, all plots present a specific amount of contention that is significantly more frequent, which determines the peak of the distribution. Moreover, all worst-case delays approach the values predicted in Table 7.

2) SYNCHRONIZATION PERFORMANCE

Next, we evaluate the max|TE| performance achieved under the given delay distributions. Unlike in Section VII-A, we avoid drawing a direct comparison between the experiments with the XO and the OCXO. It would be unfair to compare them directly due to the different delay distributions between their realizations (show in Fig. 12).

In both experiments discussed in the sequel (with the XO and OCXO), we adopt the maximum operator in (47) to obtain drift estimates. We have verified that this choice leads to better drift estimation performance, given that maximally-delayed m-to-s packets are more likely than minimally-delayed in these experiments. This probability difference is noticeable in the m-to-s delays shown in Fig. 13. Note there are a few low realizations close to 19.75 μs, whereas the realizations close to the maximum are very frequent.

Fig. 14 shows the max|TE| results with the XO over four hops. For clarity, it omits the raw time offset measurements from (6), whose max|TE| is mostly above 1200 ns. Similar to previous results, the static delay asymmetry is the predominant error component in Fig. 14a (without bias correction).

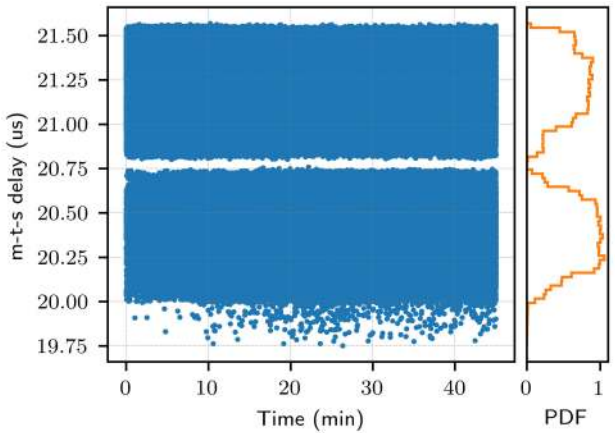


FIGURE 13. m-to-s PTP packet delays and the corresponding PDF on the experiment with CBR BG traffic over 4 hops using the OCXO.

According to Table 8, the least asymmetric delay metric in this experiment is observed between the minimum m-to-s and s-to-m delays. Hence, the sample-minimum approach achieves the best max|TE| performance in Fig. 14a. Furthermore, the max|TE| curves are reasonably close to their corresponding static asymmetries in Table 8 (their absolute values). That is, LS, KF, and sample-average approach the average asymmetry, sample-maximum is near the maximum asymmetry, and so on. These results imply that the estimators are effectively producing low-variance estimates.

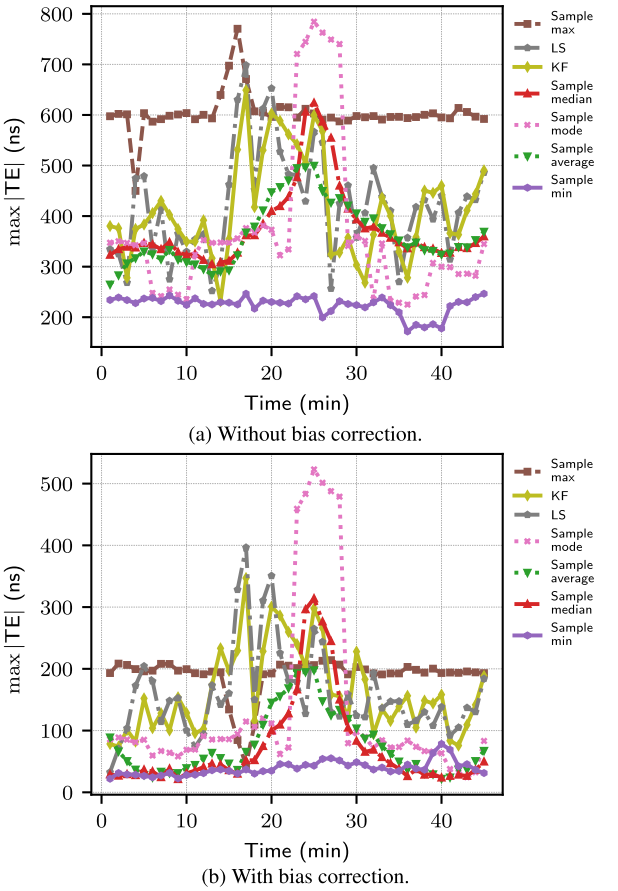
TABLE 8. Static delay asymmetry over four hops with CBR BG traffic on the acquisitions with the XO and the OCXO.

Operator	Average	Min	Max	Median
XO: Asymmetry (ns)	−301	−216	−751	−310
OCXO: Asymmetry (ns)	−925	−537	−962	−1187

In this experiment, the maximum collision between the s-to-m packets was mostly stable over 45 min, aside from around minute 15 of the experiment, where the collision-induced queuing delays increased by roughly 350 ns, as shown in Fig. 15. Correspondingly, at this point, the asymmetry changed by roughly −175 ns. Hence, in reality, the maximum asymmetry was around −576 ns most of the time, whereas Table 8 shows the global asymmetries.

In Fig. 14a, it is noteworthy also that the sample-mode produces more reasonable results than in the absence of BG traffic, although still with large fluctuations. The reason is that the delay distributions (top right corner in Fig. 12) present prominent peaks that are beneficial for sample-mode's operation. Furthermore, note that in this experiment there is no significant advantage to using the model-based LS and KF approaches. The sample-minimum outperformed them because it experienced lower asymmetry.

Next, Fig. 14b shows the results with bias correction based on the values in Table 8. Even with bias correction, the sample-minimum approach still yields one of the best



(a) Without bias correction.

(b) With bias correction.

FIGURE 14. max|TE| results with CBR BG traffic over 4 hops using the XO.

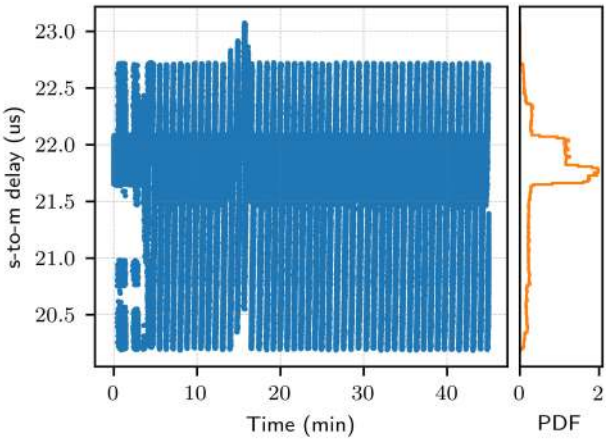


FIGURE 15. s-to-m PTP packet delays and the corresponding PDF on the experiment with the XO over four hops.

results. The rationale is that this estimator reduces the variance very well. The s-to-m delays shown in Fig. 15 present a stable minimum over the acquisition. Hence, as long as the observation window lengths are long enough to capture the moments when the s-to-m collisions are lower (which happens almost periodically due to the packet generation timing), the sample-minimum filtering can clean up the variance successfully.

In this experiment, a window of 64 seconds consistently covers the nearly periodic low values of s-to-m delays. This is confirmed in Fig. 16, which shows the global max|TE| of each algorithm for varying window lengths. Note that, once the window length becomes 2^{13} samples (i.e., 64 secs), the sample-minimum performance improves significantly. Furthermore, as the window increases to cover multiple periods of the delays in Fig. 15, the performance improves further.

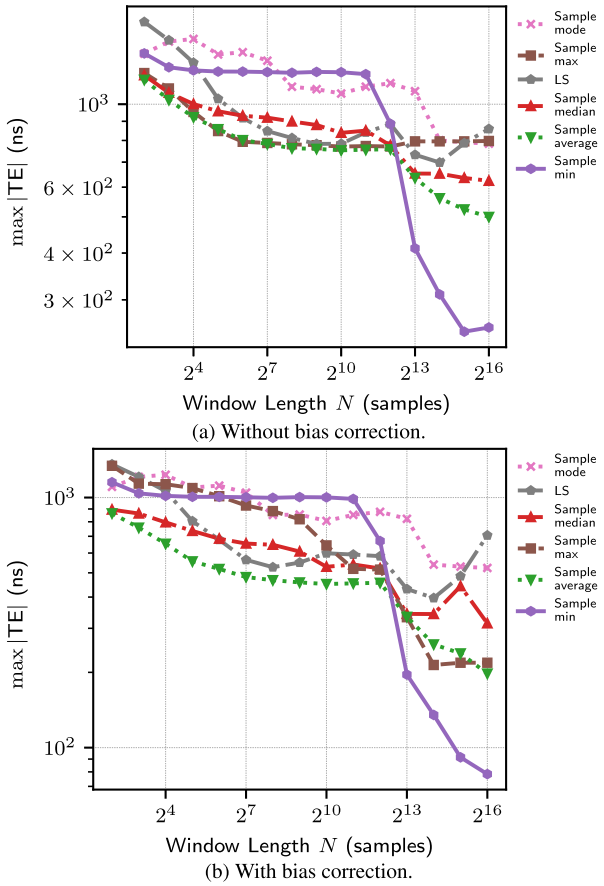


FIGURE 16. max|TE| performance for varying observation window lengths under CBR BG traffic with four network hops using the XO.

In Fig. 14b, the sample-minimum's max|TE| remains mostly around 40 ns with a peak of 80 ns. This accuracy is comparable to the one achieved without BG traffic, as demonstrated in Fig. 10c. Moreover, the sample-maximum strategy has the potential to perform similarly to the sample-minimum, given that the maximum delays are also stable over the experiment, as noticeable in Fig. 15. However, our bias correction module corrects the bias based on the global asymmetry in Table 8, which deviated from the most frequent asymmetry due to the abnormally high collision around minute 15. In a more advanced bias correction implementation, this problem could be avoided, e.g., by detecting and ignoring delay outliers.

Next, Fig. 17 shows the max|TE| results in the experiment with the OCXO over four hops while omitting the raw time offset measurements in Fig. 17b for better visibility. In this

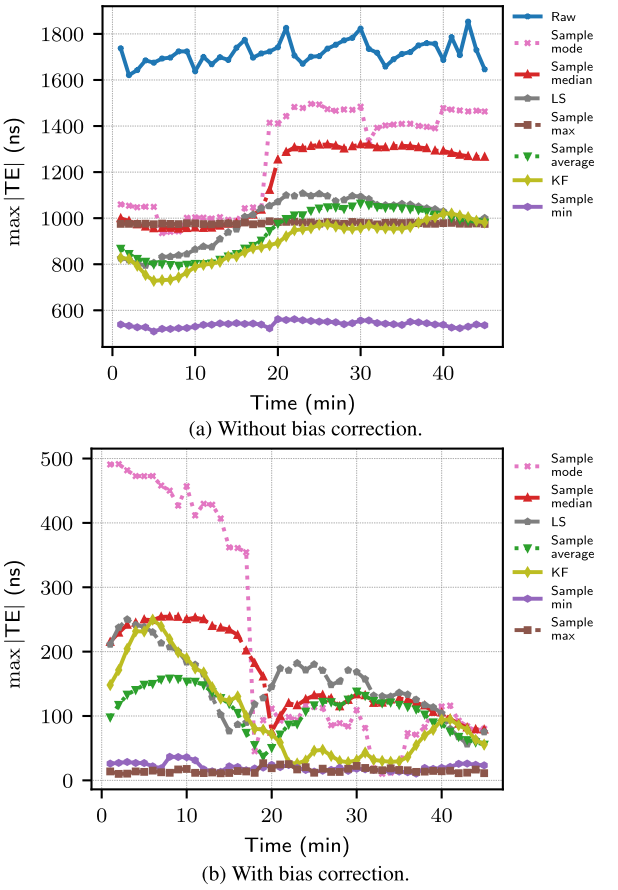


FIGURE 17. max|TE| results with CBR BG traffic over 4 hops using the OCXO.

experiment, the s-to-m delays (bottom right corner in Fig. 12) experience peak collisions close to the worst-case delays predicted by (53). Correspondingly, the static delay asymmetries (summarized in Table 8) are relatively high, given that they include the average queuing delays.

Among the asymmetries in Table 8, the minimum metric is again the least asymmetric. Correspondingly, the sample-minimum filtering achieves the best max|TE| performance in Fig. 17a (without bias correction). Furthermore, some max|TE| curves in Fig. 17 are more stable around a fixed value than in Fig 14. For example, the sample-maximum curve is almost constant both with and without bias correction. The reason stems from the stability of the delays in this acquisition. With the OCXO, the timing relationship between PTP and BG packets generated by the FPGAs and their corresponding contention patterns become more stable. As a result, the s-to-m delays (shown in Fig. 18) do not include any outliers and nearly periodically vary over the same levels.

With such a consistent delay pattern with frequent minimum and maximum values, both the sample-minimum and sample-maximum strategies produce very low-variance estimates. Hence, the max|TE| is again primarily determined by the biases. Furthermore, because the contention-free m-to-s delays (shown in Fig. 13) are also very stable, the drift

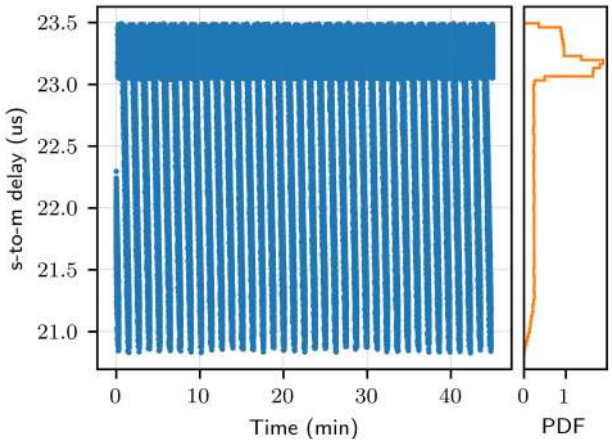


FIGURE 18. s-to-m PTP packet delays and the corresponding PDF on the experiment with CBR BG traffic over 4 hops using the OCXO.

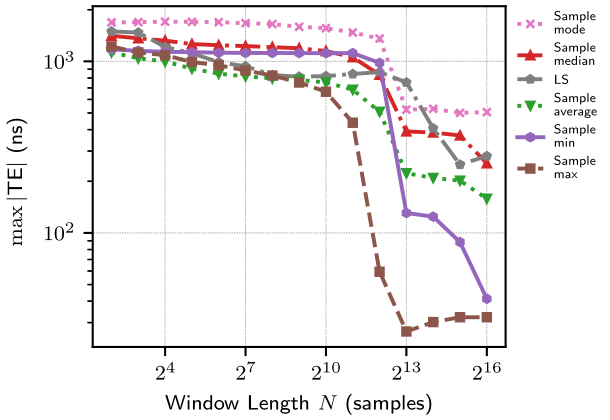


FIGURE 19. max|TE| performance for varying window lengths under CBR BG traffic, with four network hops, using the OCXO, and with bias correction.

estimations using (46) and (47) perform very well. Consequently, the drift-compensation step in (23) allows large observation windows for packet filtering.

With all these effects combined, the algorithms achieve excellent max|TE| with bias correction in Fig. 17b. In particular, the sample-maximum strategy remains consistently below a remarkably low max|TE| of 20 ns, and the sample-minimum performs similarly with slightly worse peak max|TE|. This performance is roughly 60 times better than the max|TE| derived from the bias-corrected version of the raw time offset measurements from (6). Also, these accuracy levels are close to the ones achieved without BG traffic (shown in Fig. 10d).

In Fig. 17, there was no significant advantage in using the model-based LS and KF strategies. In this particular case, KF does approach the best performance marks, but not consistently throughout the experiment. The intuition is that, in this case, the LS and KF approaches provide less effective mechanisms to overcome the PDV.

Lastly, Fig. 19 shows the global max|TE| for varying window lengths in the scenario of Fig. 17b, i.e., with bias correction. Similarly to the experiment with the XO, this plot reveals that a window length of at least 2^{13} samples (or 64 seconds) is necessary to cover the delay statistics, especially the full excursion of the nearly periodic s-to-m delays. This 64-second window was optimal for sample-maximum, whereas the sample-minimum, sample-median, and sample-average estimators performed best with the longest evaluated window of 512 seconds. Besides, note that, unlike in the results without BG traffic (Fig. 11) and the results of [18], the LS performance is not a predictable convex-shaped curve as a function of the window length. The PDV alters this behavior in a way that is hard to anticipate. The same holds in Fig. 16, in the experiment with the XO.

C. PTP UNDER VBR BG TRAFFIC

Finally, we analyze the performance under the VBR BG cross-traffic described in Section VI-C. In this experiment, we generate each VBR stream with a peak rate of 75 Mbps so that each VBR traffic host generates and consumes up to 150 Mbps concurrently. Overall, this leads to a cross-traffic utilization below 10% over the links shared with PTP traffic, where each link transports up to 75 Mbps bidirectional BG traffic. Hence, this evaluation focuses on the PTP performance under a PTP-unaware network with light cross-traffic load. Meanwhile, for brevity, other cross-traffic scenarios with heavier utilization are left for future investigations. Moreover, we restrict our analysis to a single acquisition using the OCXO and with bias correction.

With cross-traffic, both the m-to-s and s-to-m delays can experience contention. Fig 20 shows the m-to-s delays and their corresponding cumulative distribution function (CDF). Compared to the previous experiments with in-line BG traffic, Fig 20 shows significantly higher delays, primarily from the collisions between PTP packets and the large BG (1518 bytes) packets that compose Network Traffic Model 2 [57]. Nevertheless, due to the light network load, the vast majority of the delays are concentrated at low values, as confirmed by the CDF. The same holds in the s-to-m direction, which presents a similar delay distribution. Hence, similar to the scenarios discussed in [19], one can expect that the sample-minimum filtering succeeds in this scenario. Based on this expectation, we adopt the minimum operator in (47) to obtain time offset drift estimates in this experiment.

Fig. 21 shows the max|TE| performance in this scenario, including bias correction. For better visualization, it omits the sample-maximum curves, which performs poorly (max|TE| up to nearly 5 μs) due to the sporadic delay peaks that are evident in Fig. 20. It also omits the sample-mode, whose max|TE| peaks at nearly 375 ns. Furthermore, it omits the raw measurements from (6), which peaked at 15 μs. Note that the sample-minimum and sample-median strategies present excellent performances, both consistently under a max|TE| of 20 ns. The sample-average follows closely because most delay realizations consist of low values, as indicated by the

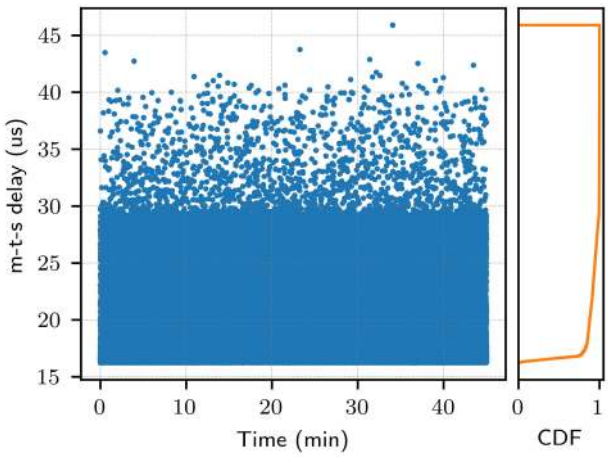


FIGURE 20. m-to-s PTP delays and the corresponding CDF on the experiment with VBR BG traffic.

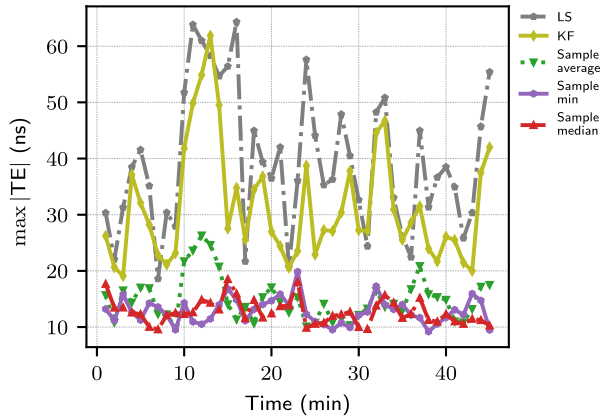


FIGURE 21. max|TE| results with VBR BG traffic over 4 hops using the OCXO.

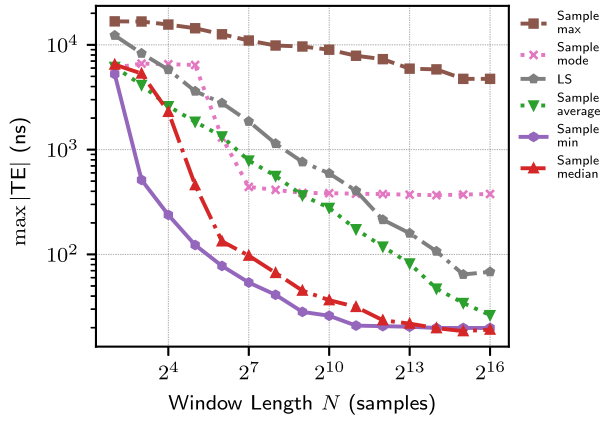


FIGURE 22. max|TE| performance for varying window lengths under VBR BG traffic, with four network hops, using the OCXO, and with bias correction.

CDF in Fig. 20. For the same reason, the LS and KF strategies perform reasonably well, although slightly worse than the other algorithms in Fig. 21.

As in previous experiments, the results in Fig. 21 are based on optimized configurations defined by PTP-DAL. For instance, Fig 22 shows the global max|TE| as a function of the observation window length in this experiment. Note that most algorithms favor relatively long observation windows. More specifically, all algorithms perform better with windows between 2^{14} and 2^{16} samples, i.e., between 128 and 512 seconds. Besides, even though PTP-DAL optimizes the window lengths independently for each dataset, we have observed similar window tuning results by replicating this experiment under the same conditions. The rationale is that the PTP delay distributions observed with VBR cross-traffic are reasonably repeatable from experiment to experiment.

VIII. CONCLUSION

As network-based clock synchronization becomes increasingly pervasive in numerous modern technology domains, it is often attractive to deploy PTP synchronization over timing-unaware networks. Nevertheless, such networks commonly present a harsh and less understood environment for synchronization performance. Thus, current high accuracy applications typically require timing-aware networks.

In this work, we aimed to expand the frontier of understanding regarding PTP-based synchronization algorithms for timing-unaware networks. We presented an in-depth survey and analysis of two main classes of algorithms: packet filtering strategies that rely on observation windows with constant time offset and estimators that incorporate the oscillator model (LS and KF). We proposed a window-based filtering formulation involving time offset drift compensation before the operators applied on each observation window and discussed the main aspects of the chosen LS and KF formulation. We also described relevant frequency offset estimation strategies and, in particular, how the frequency offset estimates are used to compensate the time offset drift within the filtered observation windows.

Subsequently, we described the FPGA-based testbed and the open-source PTP-DAL software developed in-house for reproducible experiments. For instance, we highlighted how the time offset drift estimation, bias correction, and time offset estimation modules are arranged on PTP-DAL's processing architecture. In terms of hardware, we highlighted the design choices adopted to enable the acquisition of timestamp datasets containing truth metrics (i.e., labels), such as the dual-RTC architecture implemented on the slave clocks.

More importantly, this work presented a reliable comparison of synchronization algorithms under varying BG traffic and oscillator stability scenarios. Unlike the vast majority of the related literature, this work's comparison was based on timestamps acquired from real hardware, with real oscillators and networking conditions. Furthermore, the experiments showed delay distributions significantly distinct to the theoretical distributions considered and simulated in other works.

In the experimental section, this work demonstrated the importance of the observation window length used in many synchronization algorithms. In each experiment, we analyzed

the performance over varying window lengths and showed that the best performing windows were mostly in the order of a few minutes. In cases with periodic delay patterns, it was critical to use windows large enough to cover the full delay periodicity. In this context, the drift compensation processing proposed for window-based algorithms was essential to allow their operation with long windows. Furthermore, it was critical to estimate the time offset drifts accurately despite the intense PDV-induced noise.

The experiments also highlighted that the delay asymmetry commonly represents the predominant source of time synchronization error when the estimators are well-tuned to produce low-variance estimates. In some cases, this asymmetry was dominated by the queue-induced asymmetry, which can be overcome by strategies beyond this work's scope. In other cases, it was mainly composed of static hardware asymmetries, which can be calibrated, for example, in the discussed APTS scenario. Ultimately, we showed that, when the bias induced by the delay asymmetry is appropriately corrected, there are often one or more estimators that produce high-quality time offset estimates.

Some experiments showed exceptional performance levels. For example, on a 45 min experiment using an OCXO on the slave clock on a network with 95% in-line BG traffic load, the sample-maximum algorithm could maintain the max|TE| below 20 ns. In another scenario with light cross-traffic load, the sample-minimum and sample-median algorithms achieved similar performance levels. Both experiments succeeded in approaching the performance demonstrated in a scenario without BG traffic, i.e., with a timing-unaware network dedicated to PTP traffic.

In the experiments without bias correction, the best-performing algorithms were the ones experiencing lower static delay asymmetry. Furthermore, in all experiments with significant PDV due to BG traffic, there was no significant advantage to using the LS and KF estimators incorporating the oscillator model. Instead, the window-based filtering strategies such as sample-minimum and sample-maximum were more effective in overcoming the PDV noise. Finally, the results achieved with the OCXO were generally superior to the results achieved with the XO, as expected.

Future works shall explore schemes for parameter tuning on practical scenarios where the slave does not know the residual time synchronization error resulting from each algorithm configuration. In scenarios where PTP is a secondary timing source, one can explore schemes to tune and select the more appropriate PTP synchronization algorithms when the primary timing source is active. Moreover, future works shall explore the labeled datasets discussed in this work to analyze other algorithms, such as machine learning strategies, as well as more advanced combinations of the investigated algorithms. Besides, other topics to be addressed in future works include the comparison of algorithms in terms of computational cost and investigations regarding the performance levels achieved under other varieties of network topologies and packet delay characteristics.

REFERENCES

[1] S. Ruffini, S. Rodrigues, M. Lipinski, and J.-C. Lin, "Synchronization standards toward 5G," *IEEE Commun. Standards Mag.*, vol. 1, no. 1, pp. 50–51, Mar. 2017.

[2] N. Moreira, J. Lazaro, U. Bidarte, J. Jimenez, and A. Astarloa, "On the utilization of system-on-chip platforms to achieve nanosecond synchronization accuracies in substation automation systems," *IEEE Trans. Smart Grid*, vol. 8, no. 4, pp. 1932–1942, Jul. 2017.

[3] T. Ahmed, S. Rahman, M. Tornatore, K. Kim, and B. Mukherjee, "A survey on high-precision time synchronization techniques for optical datacenter networks and a zero-overhead microsecond-accuracy solution," *Photonic Netw. Commun.*, vol. 36, no. 1, pp. 56–67, Aug. 2018.

[4] Z. Idrees, J. Granados, Y. Sun, S. Latif, L. Gong, Z. Zou, and L. Zheng, "IEEE 1588 for clock synchronization in industrial IoT and related applications: A review on contributing technologies, protocols and enhancement methodologies," *IEEE Access*, vol. 8, pp. 155660–155678, 2020.

[5] J. Lopez-Jimenez, J. L. Gutierrez-Rivas, E. Marin-Lopez, M. Rodriguez-Alvarez, and J. Diaz, "Time as a service based on white rabbit for finance applications," *IEEE Commun. Mag.*, vol. 58, no. 4, pp. 60–66, Apr. 2020.

[6] K. B. Stanton, "Distributing deterministic, accurate time for tightly coordinated network and software applications: IEEE 802.1AS, the TSN profile of PTP," *IEEE Commun. Standards Mag.*, vol. 2, no. 2, pp. 34–40, Jun. 2018.

[7] M. Levesque and D. Tipper, "A survey of clock synchronization over packet-switched networks," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 4, pp. 2926–2947, 4th Quart., 2016.

[8] *IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*, Standard 1588-2008, Jul. 2008, pp. 1–300.

[9] *IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*, Standard 1588-2019 (Revision IEEE Std 1588-2008), 2020, pp. 1–499.

[10] F. Girela-Lopez, J. Lopez-Jimenez, M. Jimenez-Lopez, R. Rodriguez, E. Ros, and J. Diaz, "IEEE 1588 high accuracy default profile: Applications and challenges," *IEEE Access*, vol. 8, pp. 45211–45220, 2020.

[11] *Precision Time Protocol Telecom Profile for Phase/Time Synchronization With Full Timing Support From the Network*, document ITU-T, Rec. G.8275.1, Mar. 2020.

[12] *Timing Characteristics of Synchronous Ethernet Equipment Slave Clock (EEC)*, document ITU-T, Rec. G.8262, Nov. 2018.

[13] *Precision time Protocol Telecom Profile for Phase/Time Synchronization With Partial Timing Support From the Network*, document ITU-T, Rec. G.8275.2, Mar. 2020.

[14] T. Pearson and K. Shenoi, "A case for assisted partial timing support using precision timing protocol packet synchronization for LTE–A," *IEEE Commun. Mag.*, vol. 52, no. 8, pp. 136–143, Aug. 2014.

[15] "What is partial timing support (PTS)," Calnex Solutions Ltd., Linlithgow, U.K., White Paper CX6006, 2020. [Online]. Available: https://info.calnexsol.com/acton/attachment/28343/f-f90192cf-d2a7-4685-89d6-01d94d118694/1/-/-/-/-/CX6006v1.0%20-%20What%20is%20Partial%20Timing%20Support.pdf

[16] *Network Limits for Time Synchronization in Packet Networks With Partial Timing Support From the Network*, document ITU-T, Rec. G.8271.2, Aug. 2017.

[17] *Timing Characteristics of Telecom Boundary Clocks and Telecom Time Slave Clocks for Use With Partial Timing Support From the Network*, document ITU-T, Rec. G.8273.4Mar. 2020.

[18] G. Cena, S. Scanzio, and A. Valenzano, "Reliable comparison of clock discipline algorithms for time synchronization protocols," in *Proc. IEEE 20th Conf. Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2015, pp. 1–9.

[19] I. Hadzic and D. R. Morgan, "On packet selection criteria for clock recovery," in *Proc. Int. Symp. Precis. Clock Synchronization Meas., Control Commun.*, Oct. 2009, pp. 1–6.

[20] I. Hadzic and D. R. Morgan, "Adaptive packet selection for clock recovery," in *Proc. IEEE Int. Symp. Precis. Clock Synchronization Meas., Control Commun.*, Sep. 2010, pp. 42–47.

[21] M. Anyaegbu, C.-X. Wang, and W. Berrie, "A sample-mode packet delay variation filter for IEEE 1588 synchronization," in *Proc. 12th Int. Conf. Telecommun.*, Nov. 2012, pp. 1–6.

[22] M. Anyaegbu, C.-X. Wang, and W. Berrie, "Dealing with packet delay variation in IEEE 1588 synchronization using a sample-mode filter," *IEEE Intell. Transp. Syst. Mag.*, vol. 5, no. 4, pp. 20–27, winter 2013.

[23] B. Noseworthy, "Direct measurement of ingress and egress latency on 1000Base-T gigabit Ethernet links," in *Proc. IEEE Int. Symp. Precis. Clock Synchronization for Meas., Control, Commun. (ISPCS)*, Sep. 2014, pp. 53–58.
[24] N. Simanic, R. Exel, P. Loschmidt, T. Bigler, and N. Kero, "Compensation of asymmetrical latency for Ethernet clock synchronization," in *Proc. IEEE Int. Symp. Precis. Clock Synchronization Meas., Control Commun.*, Sep. 2011, pp. 19–24.
[25] H. Peek and P. Jansweijer, "White rabbit absolute calibration," in *Proc. IEEE Int. Symp. Precis. Clock Synchronization for Meas., Control, Commun. (ISPCS)*, Sep. 2018, pp. 1–5.
[26] S. Lee, "An enhanced IEEE 1588 time synchronization algorithm for asymmetric communication link using block burst transmission," *IEEE Commun. Lett.*, vol. 12, no. 9, pp. 687–689, Sep. 2008.
[27] S. Lee, S. Lee, and C. Hong, "An accuracy enhanced IEEE 1588 synchronization protocol for dynamically changing and asymmetric wireless links," *IEEE Commun. Lett.*, vol. 16, no. 2, pp. 190–192, Feb. 2012.
[28] *Time and Phase Synchronization Aspects of Telecomunication Networks*, document ITU-T, Rec. G.8271, Aug. 2017.
[29] Z. Chaloupka, N. Alsindi, and J. Aweya, "Clock synchronization over communication paths with queue-induced delay asymmetries," *IEEE Commun. Lett.*, vol. 18, no. 9, pp. 1551–1554, Sep. 2014.
[30] "Latency in Ethernet switches," Plexxi, Nashua, NH, USA, White Paper 2016. [Online]. Available: https://networking.report/Resources/Whitepapers/196c5828-cb30-4664-8269-c524c18b55eb_Latency-in-Ethernet-Switches.pdf
[31] A. K. Karthik and R. S. Blum, "Robust clock skew and offset estimation for IEEE 1588 in the presence of unexpected deterministic path delay asymmetries," *IEEE Trans. Commun.*, vol. 68, no. 8, pp. 5102–5119, Aug. 2020.
[32] M. Levesque and D. Tipper, "Improving the PTP synchronization accuracy under asymmetric delay conditions," in *Proc. IEEE Int. Symp. Precis. Clock Synchronization Meas., Control, Commun. (ISPCS)*, Oct. 2015, pp. 88–93.
[33] M. J. Hajikhani, T. Kunz, and H. Schwartz, "A recursive method for clock synchronization in asymmetric packet-based networks," *IEEE/ACM Trans. Netw.*, vol. 24, no. 4, pp. 2332–2342, Aug. 2016.
[34] B. Mochizuki and I. Hadzic, "Improving IEEE 1588v2 clock performance through controlled packet departures," *IEEE Commun. Lett.*, vol. 14, no. 5, pp. 459–461, May 2010.
[35] I. Freire, I. Sousa, P. Bemerguy, A. Klautau, I. Almeida, C. Lu, and M. Berg, "Analysis of controlled packet departure to support Ethernet fronthaul synchronization via PTP," in *Proc. IEEE Int. Symp. Precis. Clock Synchronization Meas., Control, Commun. (ISPCS)*, Sep. 2018, pp. 1–6.
[36] A. Bletsas, "Evaluation of Kalman filtering for network time keeping," *IEEE Trans. Ultrason., Ferroelectr., Freq. Control*, vol. 52, no. 9, pp. 1452–1460, Sep. 2005.
[37] H. Puttnies, P. Danielis, and D. Timmermann, "PTP-LP: Using linear programming to increase the delay robustness of IEEE 1588 PTP," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2018, pp. 1–7.
[38] H. Puttnies, E. Schweissguth, D. Timmermann, and J. Schacht, "Clock synchronization using linear programming, multicasts, and temperature compensation," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2019, pp. 1–6.
[39] G. Giorgi and C. Narduzzi, "Performance analysis of Kalman-filter-based clock synchronization in IEEE 1588 networks," *IEEE Trans. Instrum. Meas.*, vol. 60, no. 8, pp. 2902–2909, Aug. 2011.
[40] G. Cena, S. Scanzio, A. Valenzano, and C. Zunino, "Implementation and evaluation of the reference broadcast infrastructure synchronization protocol," *IEEE Trans. Ind. Informat.*, vol. 11, no. 3, pp. 801–811, Jun. 2015.
[41] G. Giorgi, "An event-based Kalman filter for clock synchronization," *IEEE Trans. Instrum. Meas.*, vol. 64, no. 2, pp. 449–457, Feb. 2015.
[42] A. K. Karthik and R. S. Blum, "Optimum full information, unlimited complexity, invariant, and minimax clock skew and offset estimators for IEEE 1588," *IEEE Trans. Commun.*, vol. 67, no. 5, pp. 3624–3637, May 2019.
[43] A. K. Karthik and R. S. Blum, "Estimation theory-based robust phase offset determination in presence of possible path asymmetries," *IEEE Trans. Commun.*, vol. 66, no. 4, pp. 1624–1635, Apr. 2018.
[44] G. Cena, S. Scanzio, and A. Valenzano, "A neural network clock discipline algorithm for the RBIS clock synchronization protocol," in *Proc. 14th IEEE Int. Workshop Factory Commun. Syst. (WFCS)*, Jun. 2018, pp. 1–10.
[45] *Definitions and Terminology for Synchronization Networks*, document ITU-T, Rec. G.810, Aug. 1996.
[46] D. W. Allan, "Time and frequency (time-domain) characterization, estimation, and prediction of precision clocks and oscillators," *IEEE Trans. Ultrason., Ferroelectr., Freq. Control*, vol. 34, no. 6, pp. 647–654, Nov. 1987.
[47] I. Hadvzic, D. R. Morgan, and Z. Sayeed, "A synchronization algorithm for packet MANs," *IEEE Trans. Commun.*, vol. 59, no. 4, pp. 1142–1153, Apr. 2011.
[48] A. Guruswamy, R. S. Blum, S. Kishore, and M. Bordogna, "Minimax optimum estimators for phase synchronization in IEEE 1588," *IEEE Trans. Commun.*, vol. 63, no. 9, pp. 3350–3362, Sep. 2015.
[49] S. M. Kay, *Fundamentals of Statistical Signal Processing, Estimation Theory*, vol. 1, 1st ed. Upper Saddle River, NJ, USA: Prentice-Hall, Apr. 1993.
[50] J. Rutman and F. L. Walls, "Characterization of frequency stability in precision frequency sources," *Proc. IEEE*, vol. 79, no. 7, pp. 952–960, Jul. 1991.
[51] *Definitions and Terminology for Synchronization in Packet Networks*, document ITU-T, Rec. G.8260, Mar. 2020.
[52] I. Freire, C. Lu, M. Berg, and A. Klautau, "An FPGA-based design of a packetized fronthaul testbed with IEEE 1588 clock synchronization," in *Proc. Eur. Wireless, 23th Eur. Wireless Conf.*, May 2017, pp. 1–6.
[53] I. Freire, I. Almeida, E. Medeiros, M. Berg, C. Lu, E. Trojer, and A. Klautau, "Testbed evaluation of distributed radio timing alignment over Ethernet fronthaul networks," *IEEE Access*, vol. 8, pp. 87960–87977, 2020.
[54] T. Włostowski, "Precise time and frequency transfer in a white rabbit network," Ph.D. dissertation, Dept. Electron. Inf. Technol., Inst. Radioelectron., Warsaw Univ. Technol., Warsaw, Poland, 2011.
[55] C. Mallela, K. Tholu, and M. Bordogna, "Timing models for PTP in Ethernet networks," in *Proc. IEEE Int. Symp. Precis. Clock Synchronization Meas., Control, Commun. (ISPCS)*, Aug. 2017, pp. 1–6.
[56] R. Greenstreet and A. Zepeda, "Improving IEEE 1588 synchronization accuracy in 1000BASE-T systems," in *Proc. IEEE Int. Symp. Precis. Clock Synchronization Meas., Control, Commun. (ISPCS)*, Oct. 2015, pp. 58–63.
[57] *Timing and Synchronization Aspects in Packet Networks*, document ITU-T, Rec. G.8261, Aug. 2019.

IGOR FREIRE (Member, IEEE) received the B.Sc. and M.Sc. degrees in electrical engineering from the Federal University of Pará (UFPA), Belém, Brazil, in 2013 and 2015, respectively, where he is currently pursuing the Ph.D. degree in electrical engineering. Since 2012, he has been with LASSE-5G and IoT Research Group, Belém. His current research interests include clock synchronization, fronthaul technologies, wireless communications, and software-defined radio.

CAMILA NOVAES was born in Pará, Brazil, in 1998. She is currently pursuing the B.S degree in computer engineering with the Federal University of Pará (UFPA), Belém, Brazil. Since 2017, she has been with LASSE-5G and IoT Research Group, Belém. Her current research interests include clock synchronization and network virtualization.

IGOR ALMEIDA (Member, IEEE) received the B.Sc. degree in computer engineering and the M.Sc. degree in electrical engineering from the Federal University of Pará (UFPA), Belém, Brazil, in 2010 and 2013, respectively, where he is currently pursuing the Ph.D. degree in electrical engineering. Since 2016, he has been with Ericsson Research, involved with topics, such as 5G, fronthaul networking, synchronization, and wireless communications.

EDUARDO MEDEIROS received the M.Sc. degree in electrical engineering from the Federal University of Pará, Belém, Brazil, in 2010, and the Licentiate and Ph.D. degrees from Lund University, Lund, Sweden, in 2015 and 2018, respectively. Since 2011, he holds a Researcher position with Ericsson Research, Stockholm, Sweden. His current research interests include signal processing for broadband communications and fronthaul for 5G systems. He was a co-recipient of two best paper awards from the IEEE International Conference on Communications and three best paper awards from the IEEE Communications Society Transmission, Access, and Optical Systems Technical Committee.

MIGUEL BERG (Senior Member, IEEE) received the B.S. degrees in electrical engineering and computer science from Mid Sweden University, Sweden, in 1995, and the Licentiate and Ph.D. degrees in wireless communication systems from the Royal Institute of Technology (KTH), Sweden, in 1999 and 2002, respectively. From 2002 to 2003, he worked with the development of 2G/3G base station antennas and tower-mounted amplifiers with Radio Components Sweden AB, and from 2004 to 2006, he held a Researcher position with KTH, in the area of wireless access. After joining Ericsson in 2007, he initially worked with xDSL line testing and G.fast, as a continuation of his work as a Researcher with Lund University, from 2006 to 2007. Since 2012, he has made significant contributions to research and development for the Ericsson Radio Dot System and targeting indoor small cells. As a Principal Researcher with Ericsson Research, Stockholm, Sweden, he is currently leading research on efficient fronthaul and lower layer splits for 4G/5G RAN. He has coauthored about 35 scientific publications (journal, letters, magazine, and conference) and 80 filed inventions.

ALDEBARO KLAUTAU (Senior Member, IEEE) received the bachelor's degree from the Universidade Federal do Pará (UFPA), in 1990, the M.Sc. degree from the Universidade Federal de Santa Catarina (UFSC), in 1993, and the Ph.D. degree in electrical engineering from the University of California at San Diego (UCSD), in 2003. He was a Visiting Scholar with Stockholm University, UCSD, and The University of Texas at Austin. He is also a Researcher with the Brazilian National Council of Scientific and Technological Development (CNPq). Since 1996, he has been with UFPA, where he is currently a Full Professor, the ITU-T TIES Focal Point, and directs LASSE. He has supervised more than 50 graduate students. He published more than 150 articles in peer-reviewed conferences and journals, and has several international patents. His research interests include machine learning and signal processing for telecommunications and embedded systems.