SSA: Microsecond-Level Clock Synchronization Based on Machine Learning for IoT Devices

Zhuochen Fan, Yanwei Xu, Peng Liu, Xiaodong Li, Ruwen Zhang, Tong Yang, *Member, IEEE*,
Wenfei Wu, *Member, IEEE*, Yuqing Li, *Associate Member, IEEE*, Li Chen,
and Gong Zhang, *Member, IEEE*

Abstract— **Clock synchronization is an essential but challenging task for internet of things (IoT) devices. The state-of-the-art data-driven Huygens solution cannot achieve accuracy for IoT networks, because the devices are usually weak in power to make massive timestamp probing for data-driven solutions. We propose the SSA clock synchronization scheme to improve the Huygens algorithm. First, SSA has a sliding window mechanism to accumulate data points for the data-driven support vector machine (SVM) algorithm in Huygens, which complements the issue of insufficient data points. Second, SSA applies a smoothing method to the periodical estimated clock offset and drift, which eliminates the noise introduced by the larger sliding window. Third, SSA makes an adaptive clock correction instead of the periodical correction in Huygens so as to avoid correcting the clock before the algorithm could stably estimate and smooth the clock offset and drift. We conduct extensive experiments on a real device (Huawei Sound X), and the results shows that our SSA can achieve synchronization accuracy of around 20 μs in the actual working environment.**

Index Terms— **Adaptive correction, clock synchronization, drift, machine learning, offset, sliding window, smoothing.**

NOMENCLATURE

Symbol	Meaning
t	Global real-world time.
Δ	Device's clock offset.
C	Device's local clock time, $C = t + \Delta$.
D	Device's clock drift.
t^s	Probe packet's sending timestamp.
t^r	Probe packet's receiving timestamp.
d	One-way delay (OWD).

I. INTRODUCTION

CLOCK synchronization plays a fundamental role in the process of data collection and transmission [1], [2], [3] for various explosively-growing internet of things (IoT) applications, such as high-speed communications [4], [5], [6], industrial automation [7], [8], [9], and smart healthcare systems [10], [11], [12], etc. Specifically, synchronized clocks can provide timestamps to decide the order of events among multiple devices, which further decides the correctness or performance of the above mechanisms. Thus, the granularity of synchronization accuracy could directly affect the atop applications and the consequent quality of user experience (QoE) [13], [14]. For example, the clock synchronization of the left and right channels of a smart speaker in a stereo scene should preferably not exceed 1ms, otherwise the human ear can discern it [15], [16], [17].

A variety of clock synchronization solutions are proposed, which can provide accuracy at different granularity in their own scenarios. For example, well-known schemes are network time protocol (NTP) [18], precision time protocol (PTP) [19], referencing broadcast synchronization (RBS) [20], time-synchronization protocol for sensor network (TPSN) [21], flooding time synchronization protocol (FTSP) [22], pair broadcast synchronization (PBS) [23], datacenter time protocol (DTP), [24], and others [10], [25], [26], while new schemes have been proposed in recent years, such as Huygens [27], data-plane time-synchronization protocol (DPTP) [28], multi-hop precision time protocol (M-PTP) [29], and others [30], [31], [32], [33], [34]. Some of them [24], [27], [28], [29], [31] can even achieve nanosecond synchronization accuracy. However, they are heavyweight, i.e., they incur a huge hardware overhead or complex deployment/implementation to work properly, which is unacceptable for common IoT devices. For specific IoT devices[1] deployed on commercial-grade hardware, the finest granularity achievable with current

---

[1] Commercial-grade IoT devices in this article mainly refer to relatively small devices built on traditional hardware platforms such as microprocessors or microcontrollers [35], etc.

clock synchronization efforts, as far as we know, is only on the order of milliseconds (e.g., [7], [33], [34], [36], [37], [38]).

Among the clock synchronization solutions, data-driven algorithms combine the massive data measurement and data analytic or machine learning methods to improve the synchronization accuracy. The Huygens algorithm [27] is a representative solution that provides the state-of-the-art most fine-grained accuracy of tens of to hundreds of nanoseconds. The Huygens algorithm periodically measures massive data points of probing timestamps between devices, applies support vector machine (SVM) [39] to the data within each period to estimate the clock offset and clock drift, and corrects the clock according to the estimated clock offset and drift. Such a design has shown its success in the device synchronization in a data center environment.

However, IoT devices' properties prevent the data-driven approach to be applied. Essentially, the IoT devices usually have insufficient computation/communication capabilities to make massive measurements, but also the wireless environment and software measurement component could be rather noisy; thus, it is challenging to apply data-driven methods to eliminate noises in IoT device's measurement data. We propose the **SSA** clock synchronization scheme to overcome this challenge, which consists of three techniques: **S**liding Window, **S**moothing, and **A**daptive Correction. It inherits Huygens's periodical measurement, estimation, and correction but makes three improvements as follows: 1) it introduces a **S**liding window mechanism to complement the insufficient data points within each interval, where each window spans multiple intervals; 2) it applies a **S**moothing method to denoise the estimation among intervals; and 3) and it **A**daptively corrects device clock based on the estimation and smoothing result. Our prototypes and experiments show that in the actual working environment of the smart speaker (Huawei Sound X) with 5 GHz WiFi, the average clock offset and drift of **SSA** are 19.97 μs and 0.91 μs/s, which are 5.27 and 4.90 times better than the original Huygens, respectively. The experimental results validate the effectiveness of our three improvements/techniques.

A. Our Key Contributions

1) Our above **SSA** scheme successfully adapts the original Huygens from the data center scenario to the wireless local area network (WLAN) scenario where ordinary IoT devices generally work, and significantly promotes the synchronization granularity of ordinary IoT devices: from millisecond to microsecond level;

2) Compared with other state-of-the-art data-driven schemes, we are the first to actually implement the proposed solution from the algorithm/system level to the product level.

The remaining part of this article is organized as follows. Section II describes the latest related work, background, and motivation of clock synchronization for IoT devices. Section III describes the design of **SSA**. Section IV presents the implementation and evaluation of **SSA**. And Section V concludes this article.

II. BACKGROUND AND MOTIVATION

A. State-of-the-Arts

1) Data-Driven Methods for Clock Synchronization: In practice, synchronization protocols are likely to suffer from noise such as network fluctuations and queuing delays, which significantly reduces accuracy. And researchers tried data-driven correction methods to eliminate the noise. Huygens [27] uses SVM to remove abnormal points during synchronization and adopts the median values (hyperplane) as the measured clock offset, and further obtains the drift. SLMT [30] uses linear programming to estimate clock offset and drift, employs temperature compensation, and assumes piecewise linearity of the clock. To alleviate the delay asymmetry of the internet, the M-PTP [29] calculates the distribution of random delays based on SVM, and then utilizes the L-estimator to estimate the offset. It requires that both nodes to be synchronized must support PTP and be deployed on expensive switches.

2) Hardware/Physical Methods for Clock Synchronization: DPTP [28] implements clock synchronization on the network data-plane by designing programmable switching ASICs, but it requires expensive hardware support. Sundial [31] utilizes specialized hardware to provide a failure-tolerant periodic clock synchronization that restores back-up clocks by performing fast failure detection. Graham [32] builds a failure model based on the physical properties of the local clock and the desired synchronization accuracy, and fits enough data points collected from large-scale commodity sensors to further improve the accuracy.

3) Clock Synchronization for IoT Devices: At present, there are very few clock synchronization mechanisms specifically proposed for ordinary IoT devices, and they can almost achieve millisecond granularity. For example, some work [33], [36], [37] achieves granularity of around tens of milliseconds, while others [7], [34], [38] can achieve a synchronization fine-grained range of about 0.1–1 ms with better hardware assistance.

Summary: We enumerate some of the above state-of-the-art clock synchronization schemes for comparison in Table I, where hardware requirements refer to the hardware requirements (combined price and deployment cost, etc.) to achieve the synchronization granularity shown. Although some of them can reach the granularity of tens of nanoseconds, the hardware constraints of ordinary IoT devices do not allow them to reach such accuracy.

B. Preliminaries of Huygens

1) Clock Offset and Drift: We use the symbols in Nomenclature to elaborate the clock synchronization. IoT devices use quartz crystal oscillator to count time. When the oscillator does not work in the proper environment (e.g., temperature and humidity), its resonance frequency changes, causing inaccurate time counting. And this time counting inaccuracy is called *clock drift D*. The clock drift accumulates over time, resulting in a *clock offset* Δ. Their relationship is

$$\Delta(t) = \int_{t_0}^{t} D(x)dx + \Delta(t_0). \tag{1}$$

TABLE I
COMPARISON BETWEEN STATE-OF-THE-ART SYNCHRONIZATION SCHEMES

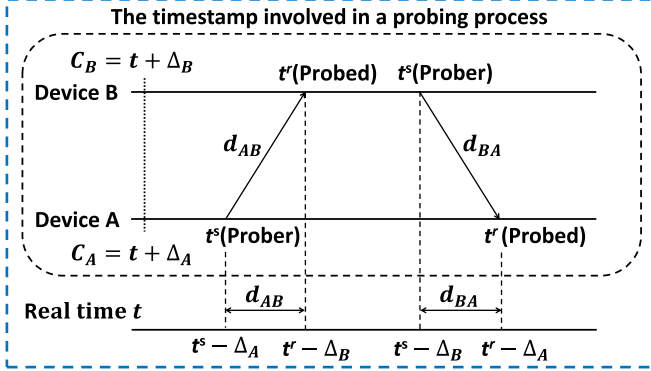| State-of-the-art Schemes	Huygens	M-PTP	DPTP	Sundial	Graham	X-Sync	**SSA** (Ours)
Synchronization Granularity	$10s \sim 100s\ ns$	$100s\ ns$	$10s \sim 100s\ ns$	$100s\ ns$	$1\mu s$	$100s\ \mu s \sim 1ms$	$20\ \mu s$
Application Scenarios	Data Center	LAN	Data Center	Data Center	Data Center	WLAN	WLAN
Hardware Requirements	High	High	High	High	High	Low	Low

Fig. 1. Derivation process of two devices exchanging timestamps to measure clock offset.

2) Timestamp Probing (Fig. 1): The clock synchronization protocol needs devices to probe each other (device-to-device). When device A probes device B, the probing packet leaves device A at A's clock time t^s and arrives at B at B's clock time t^r. Thus, we have the one-way delay (OWD) from A to B

$$d_{AB} = \left(t^r - \Delta_B\left(t^r\right)\right) - \left(t^s - \Delta_A\left(t^s\right)\right).$$

We define the relative clock offset between A and B as $\Delta_{AB} = \Delta_A - \Delta_B$. Note that $\Delta_{AB} = -\Delta_{BA}$, and Δ_{AB} varies with time due to clock drift accumulation.

We further make two assumptions: 1) t^r and t^s are close so that $\Delta_B(t^r) \approx \Delta_B(t^s)$ and 2) the OWD in the two directions between A and B are the same, i.e., $d_{AB} = d_{BA}$, denoted as d. The formula above is rewritten as

$$\left(t^s - t^r\right) = \Delta_{AB}\left(t^s\right) - d.$$

On the reverse path, each probe from B to A follows the formula:

$$\left(t^r - t^s\right) = \Delta_{AB}\left(t^s\right) + d.$$

3) Huygens Algorithm [27]: In the Huygens algorithm, each probe from A to B would generate a data point $x = t^s$ and $y = (t^s - t^r)$, and each one from B to A generates $x = t^s$ and $y = (t^r - t^s)$. The two sets of data points fall into two parts in the xy plane with a distance $2d$. Fig. 2 shows the measurement results of the two sets. In the short measurement period, $\Delta_{AB}(t^s)$ can be viewed as a straight line (i.e., clock drift $D(t)$ does not have bursty variation). Thus, the Huygens algorithm applies soft-margin SVM [40] to find the two border lines of the two sets [from A to B and from B to A in (1)] and the center line between the two border lines over a 2-s interval. The distance d can be calculated from the distance between lines, the clock offset $\Delta_{AB}(t)$ (at time t) can be calculated from the line, and the clock drift is the slope of the line.
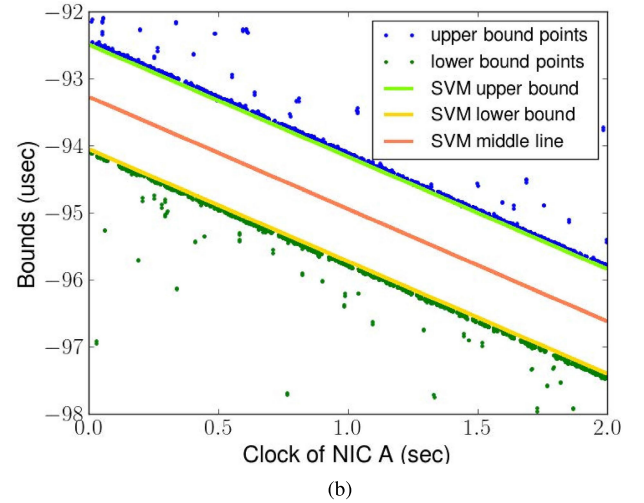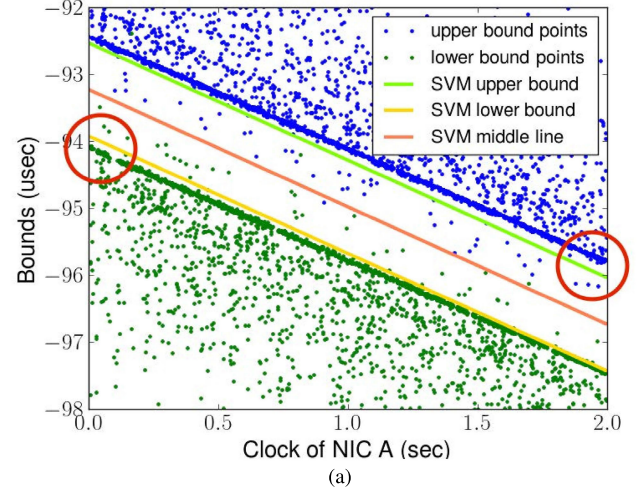

Fig. 2. SVM processing calculation diagram of the Huygens algorithm (figures taken from Geng et al. [27]). (a) Data not filtered by coded probing. (b) Data filtered by coded probing.

The Huygens algorithm has an optimization—coded probing—to eliminate noisy data points. When one device probes another one, it sends two consecutive packets and gets four device clock timestamps: 1) packet 1 with t_1^s and t_1^r and 2) packet 2 with t_2^s and t_2^r, as shown in Fig. 3. The coded probing check whether two sending timestamps and two receiving timestamps' differences $|(t_2^r - t_1^r) - (t_2^s - t_1^s)|$ is farther than a threshold ϵ, if the difference exceeds ϵ, the data points are discarded as noise; otherwise, both data points are counted in the SVM algorithm.

C. WLAN Clock Synchronization: Challenge

The synchronization of IoT devices in WLAN scenarios has two properties. First, the timestamp probing in WLAN
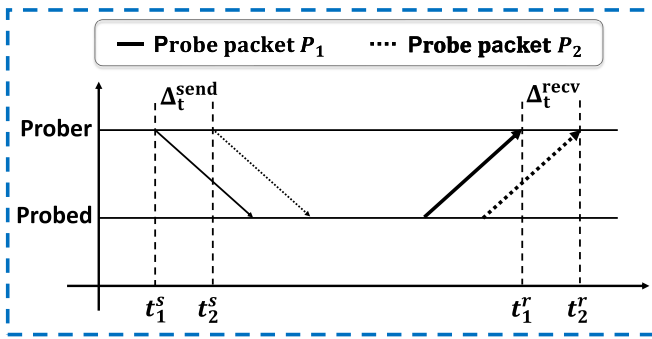
Fig. 3.   Coded probing of the Huygens algorithm.

scenarios is noisy. In the wireless channel, the probing signal can be disturbed by the channel noise; IoT devices are exposed to open environments, and their quartz crystal oscillators can be influenced by environmental condition such as temperature and humidity [26]. Inside the devices, the clock read/write are software based, which could also be disturbed by the device operating system (OS) scheduling. Second, IoT devices are weak in computation and communication [41], [42], and it is difficult for them to generate frequent and stable probing in limited time [27], [43], [44].

The two properties create a dilemma for the data-driven algorithms. Therefore, data-driven algorithms need to periodically measure timestamps and eliminate noise. But the length of the periodical measurement interval is hard to decide. If the measurement interval is too short, IoT devices cannot generate sufficient data points for the data-driven algorithm, causing the intra-interval estimation to be inaccurate; if the measurement interval is too large, the IoT device may not be in a stable state (i.e., having clock drift variation), the average estimation of a long interval cannot stand for the instantaneous estimation at the end of the interval.

D. Our Proposed Solution

We propose **SSA** to overcome the challenge. **SSA** is a combination of three improvements for Huygens—**S**liding window, **S**moothing, and **A**daptive correction. **SSA** makes periodic measurement, estimation, and correction like Huygens. To overcome the issue of insufficient data point within each interval, it introduces a sliding window mechanism to collect more data for the SVM-based estimation. The sliding window looks backward to previous intervals and accumulate sufficient data points for SVM. However, the sliding window may introduce cross-interval noise, thus **SSA** further applies data smoothing to the per-interval estimation from SVM. **SSA** applies linear regression for this smoothing.

Finally, even with the sliding window, the device still takes a while to accumulate data points in the beginning and after resetting the clock drift. With sufficient data points, per-interval SVM-based estimation can be stable. **SSA** checks the standard deviation of the clock drift estimation in past intervals, and adaptively chooses to correct the clock only when the standard deviation is small.

E. Tool: SVM

SVM is a powerful supervised learning algorithm for linear and nonlinear classification. A linear SVM is provided by the set of data points $(x_i, l_i)(1 \leq i \leq N)$, where x_i is $\mathbb{R}^2$, and l_i is a binary label, i.e., the "upper bound point" or "lower bound point" in Fig. 2. SVM classifies points with similar labels, and its goal is to find a hyperplane so that the distance from it to the nearest data point of each label is maximized.

In this article, we use SVM to "naturally" (non-artificial labeling) distinguish the two classes of data points generated by the two probing directions, namely "upper bound point" and "lower bound point." Note that these two sets of probes are on different devices, and their measurements are not shared with each other: each device measures independently and does not perform duplicated calculations. Since noise is likely to cause these two classes of data points to be mixed together, we can ensure the accuracy of SVM by tuning the SVM parameter `svmC` to denoise and separate the two classes of data points. `svmC` illustrates the tolerance/elasticity of the SVM model to misclassification (i.e., failure to distinguish two classes of data points). The larger `svmC`, the less error-tolerant the model, but overfitting may occur; the smaller `svmC`, the more error-tolerant the model, but underfitting may occur. See Section IV-B for the experiments of its impact.

III. SSA SYNCHRONIZATION ALGORITHM

A. Overall Workflow

We divide the workflow of **SSA** into three main steps: 1) time slicing and estimation of clock offset and drift; 2) smooth clock offset and drift; and 3) adaptive correction of clock frequency. The details are as follows.

1) *Step 1:* In **SSA**, the time is discretized into multiple time slices. Within each time slice, coded probing is applied to collect data points similar to Huygens (see Section II-B for details). At the end of each time slice, the sliding window mechanism is used to accumulate data points, and the SVM algorithm is applied to estimate the clock offset and drift. The raw measurement data points and the estimated clock offset and drift are stored in the database.

2) *Step 2:* Next, **SSA** smooths the estimated clock offset and drift. After the measurement and estimation of a time slice ending at time t, the algorithm fetches the estimation in previous intervals and applies linear regression [45]. In this way, smoothed offset and drift are the points on the regressed line at t, which are later stored in the database.

3) *Step 3:* Then, **SSA** computes the standard deviation of the smoothed clock drift in the current and past few intervals. If the standard deviation is small and less than a predefined threshold: it indicates that sufficient data points have been collected and the result is well denoised. Therefore, the clock is corrected; otherwise, the clock is not corrected.

In Sections III-B–III-D, we describe the three techniques corresponding to the above three steps, respectively.
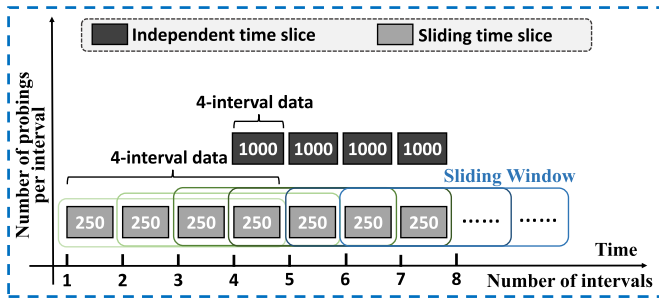
Fig. 4. Example of sliding time slices. Through sliding window technique, even if each interval has only 250 sets of data, 1000 sets of data can be provided for each processing.

Fig. 5. Process of timestamps batch-processing based on sliding time slices.

B. Sliding Window Technique

Huygens is deployed in data center networks with strong computing power. It chooses the measurement slice/interval to be 2 s, so a large number of data points can be generated in each interval. However, common IoT devices lack resources to provide such frequent probes, resulting in an insufficient number of timestamps for the SVM algorithm to guarantee measurement accuracy. For example, the Huawei Sound X [46] can only achieve 250 probes within an interval of 2 s during normal work, unless we are willing to consume a lot of computing resources to make more probes. In particular, when more than two devices are to be synchronized, more sparse data points have to be dealt with as one device (acting as a server) has to send/receive data with multiple devices at the same time.

In **SSA**, devices send probes as fast as they can, and still apply coded probing to filter noisy probes. At the end of each interval, our algorithm looks backward to a window in early intervals until the accumulated data points in the previous neighboring intervals is sufficient for the SVM algorithm. **SSA** then use SVM to compute the clock offset and drift at the end of the interval, which is called the estimated clock offset and drift.

As shown in the Fig. 4, suppose we need to provide 1000 sets of timestamp data per interval when using independent time slices. Through the sliding window technique, we only need to provide 250 sets of data in each interval. However, it is necessary to combine the data in multiple intervals in the calculation. For example, when calculating the clock offset and drift of the fourth interval in the fifth interval, the sliding window needs to use a total of 1000 sets of data for the first–fourth interval. Specifically, as shown in Fig. 5, the timestamp batch-processing based on sliding window includes the following three steps.

1) In the current time slice, k probings are performed between the probing device and the probed device, and four timestamps are generated during each probing to indicate the sending and receiving time of the packet on the two devices. The number of probings k depends on the amount of data required for SVM training and the length of the time slice. Then, we use the coded probing to filter the k groups of timestamp data, and the filter threshold is set according to the probing frequency.
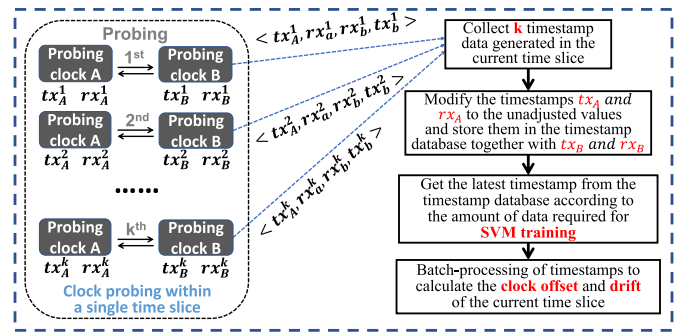
2) The timestamps collected in the current time slice should be corrected according to the cumulative clock correction value recorded in the time slice database before processing, restored to the value when the clock was not corrected, and then stored in the timestamp database.

3) Take a certain amount of timestamps from the timestamp database according to the amount of data required for training, and then use these data to calculate the clock offset Δ and drift D (*estimated clock offset and drift*), and the calculation results are stored in the time slice database after smoothing.

C. Smoothing Technique

A longer sliding window may introduce more noise, resulting in inaccurate data obtained, which may cause fluctuations and instability of the clock offset and clock drift calculated by each time slice. Therefore, **SSA** applies linear regression [45] to smooth the calculation results, i.e., use the clock offset values of the last several time slices/intervals to calculate the clock offset and drift of the current interval, which will be used as a basis for clock correction.

As shown in Fig. 6, **SSA** first obtains the clock offset $\Delta_1, \Delta_2, \ldots, \Delta_m$ of the last m time slices/intervals from the time slice database, and the start time of the time slice $t_1, t_2, \ldots, t_m$. Then, these data are linearly regression processed with the clock offset Δ_0 and the start time t_0 of the current time slice. Taking the time t of each time slice as the independent variable and the clock offset Δ as the dependent variable, linear regression is used to obtain the regressed clock offset line. Based on this, the newly smoothed clock offset Δ_0^* at the end of the current time slice is obtained, and the ratio of the regressed line D_o^* is the clock drift. Finally, **SSA** stores the smoothed data of the current time slice in the time slice database, and performs clock correction accordingly.

D. Adaptive Correction Technique

Clock offset correction refers to modifying the time of the device, and clock drift correction refers to adjust the device time counting frequency so as to compensate the drift difference between two clocks. In most solutions, the clock correction is operated periodically together with the offset and drift estimation.

In **SSA**, the clock correction is more complicated. When the system boots up, the sliding window cannot accumulate
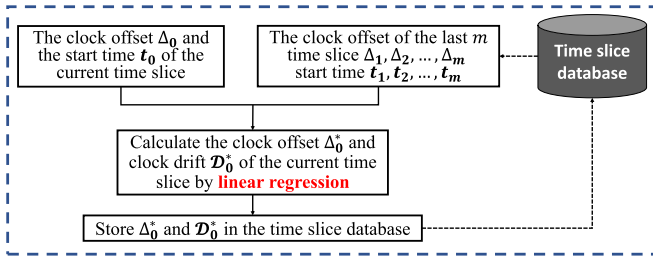
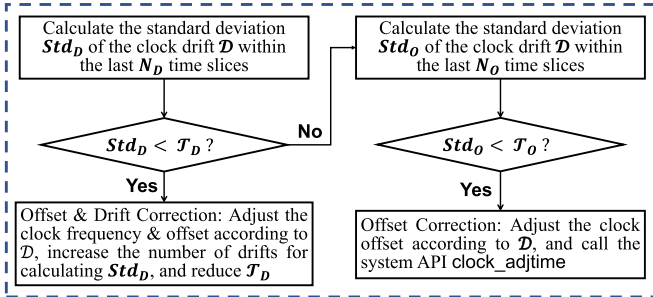Fig. 6.   Process of smoothing measurement results using linear regression.

Fig. 7.   Process of clock adaptive correction is mainly realized by flexibly setting offset threshold $\mathcal{T}_O$ and drift threshold $\mathcal{T}_D$.

Fig. 8.   Impact of the type of timestamp acquisition, where `soft` type is stable much earlier.

sufficient data points, causing the SVM and the smoothing not to be able to denoise effectively. In this case, it is not proper to correct the clock. A good intuition is to check the stability of the clock drift first (lower standard deviation means less noise), and correct the clock when the drift is stable.

Thus, the overall clock correction is as follows (Fig. 7). The algorithm checks in the standard deviation of the smoothed clock drift within a short duration and a long duration. There are three cases: 1) if the standard deviation Std_D in the short duration (N_D intervals) is smaller than a threshold $\mathcal{T}_D$ (i.e., stable), both the clock offset and clock drift are corrected; 2) otherwise, the standard deviation Std_O in the long duration (N_O intervals is checked with a threshold $\mathcal{T}_O$. If Std_O is smaller than $\mathcal{T}_O$ (i.e., relatively stable), the clock offset is corrected (but the clock drift is not); and 3) if both Std_D and Std_O exceed their thresholds, the clock is not corrected.

Note that if the clock drift is corrected, i.e., *Case 1*, the sliding window should not span intervals before drift correction. And there would not be sufficient data points problem in near future to make a window, and the algorithm is reset and warms up again. If the clock offset is correct, i.e., *Case 2*, when the sliding window still accumulates data point between intervals, but it would apply the offset correction to data points in early intervals (before the correction) so that all data points in an SVM calculation have the same time counting baseline.

IV. EVALUATION

A. Experimental Settings

1) Implementation: We directly evaluate the **SSA** synchronization algorithm in the Huawei's smart speaker product— Huawei Sound X [46]. In particular, Huawei Sound X (Linux) has a 32 bit ARMV7 with 900 MHz frequency, supports
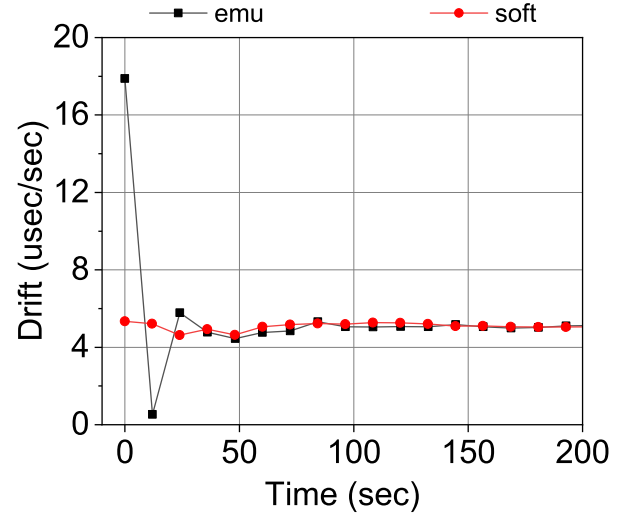
802.11 a/b/g/n/ac WLAN in 2.4/5 GHz dual frequency,[2] and Bluetooth 5.0. All code was implemented in C++. Our experimental evaluation includes **parameter tuning** and **system-synthetic experiments** (i.e., under real Huawei Sound X working conditions).

2) Evaluation Metrics: For parameter tuning, we only use **clock drift** measurements. For system-synthetic experiments, we use both **clock offset and drift** measurements.

3) Baseline: The original **Huygens** algorithm [27], disabling our **SSA** with three techniques.

B. Results of Parameter Experiments

1) Impact of the Type of Timestamp Acquisition: It also determines the type of clock adjustment. This experiment compares two types: `soft` (network-card-driven timestamp), `emu` (user mode timestamp). The difference between them is that the availability of `soft` timestamp is based on the actual situation of the network card on the host, while `emu` timestamps can be supported by any host. As shown in Fig. 8, the `soft` type is obviously more stable than the `emu` type. Therefore, we must choose `soft` type in system synthetic experiments.

2) Impact of SVM Parameter (`svmC`): It is the elastic parameter (also called penalty parameter in SVM) for calculating the clock offset. Since it is not a sensitive parameter, we take `svmC = 0.01, 0.1, 1, 10` as empirical values to better observe its impact through this experiment. As shown in Fig. 9, the drift at `svmC = 0.1` tends to stabilize at the earliest, followed by the drift at `svmC = 1`. Additionally, the drift at `svmC = 0.01` is the latest to stabilize, while the drift at `svmC = 10` fluctuates the most. Therefore, we choose `svmC = 0.1`.

---

[2]5 GHz WiFi has a larger bandwidth than 2.4 GHz WiFi and may perform more stable, because Bluetooth also has a 2.4 GHz frequency band, which may cause interference to IoT devices working under 2.4 GHz WiFi at the same time.
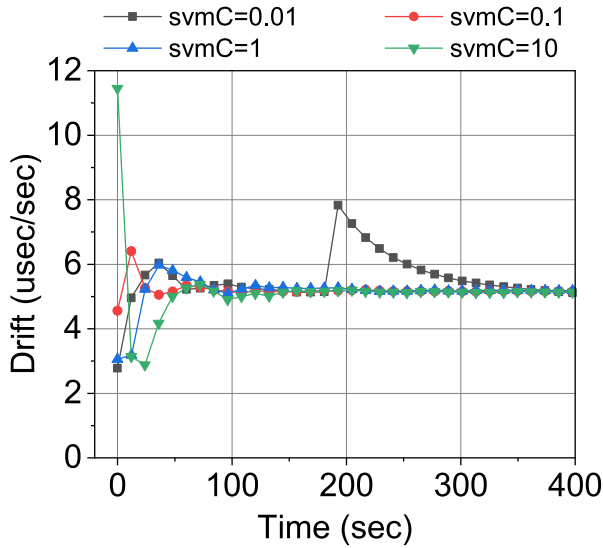
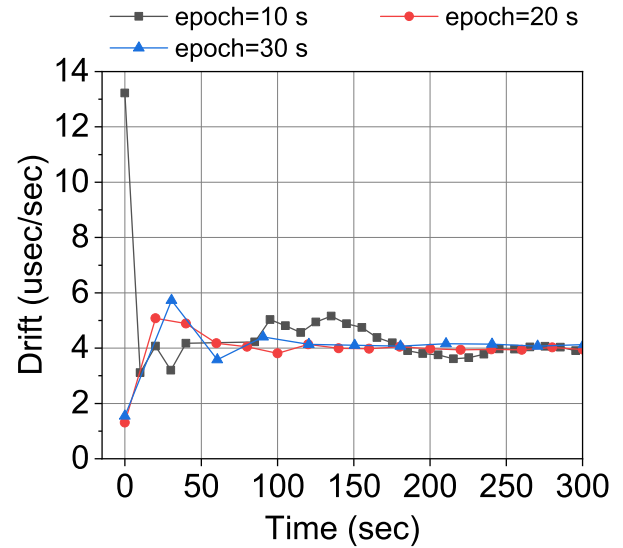Fig. 9. Impact of `svmC`, where `svmC` = 0.1 is the earliest stable.

Fig. 11. Impact of `epoch`, where `epoch` = 20 converges the earliest.
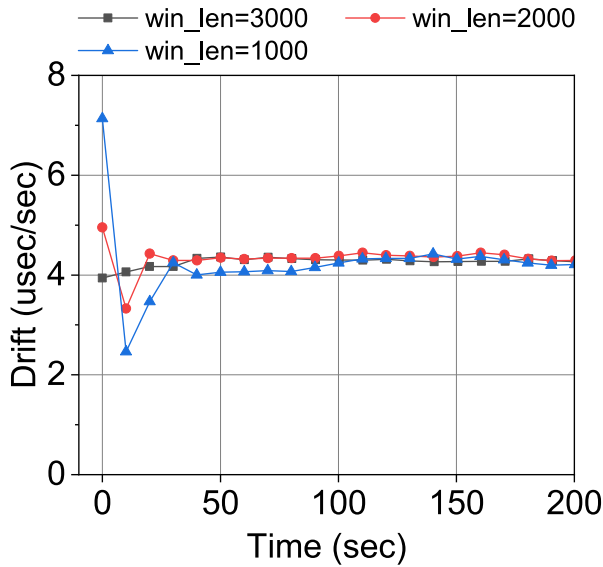

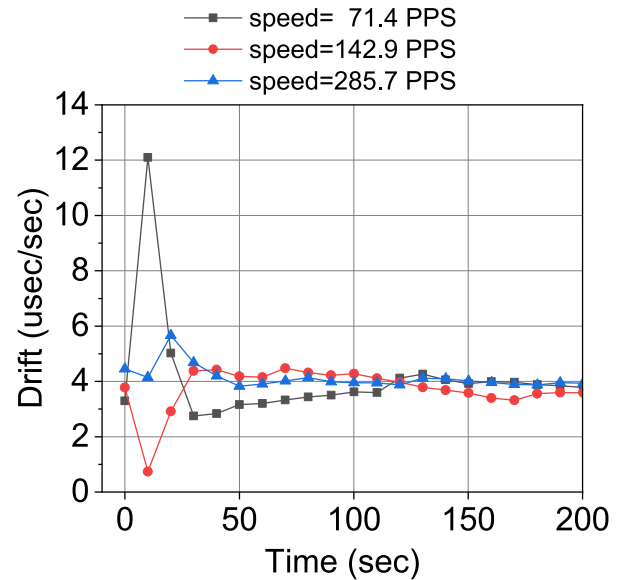Fig. 10. Impact of `win_len`, and we choose `win_len` = 2000.

Fig. 12. Impact of `speed`, and we choose `speed` = 71.4 for the reasons in Fig. 13.

3) Impact of the Length of the Sliding Window (`win_len`): It is the number of timestamps recorded. As shown in Fig. 10, when `win_len` equals 2000 or 3000, the drift is stable at the earliest. If this value is too large or too small, it will obviously affect the stability of drift because it involves packet sending speed. This will be explained later [see the analysis of (3)–(5)]. Therefore, we choose `win_len` = 2000.

4) Impact of `epoch`: It represents the length of each time slice, with the unit being seconds (*s*). This experiment is based on the fact that we must collect enough timestamps to ensure stability. As shown in Fig. 11, clock drift with `epoch` = 20 shows the earliest and most stable convergence. Therefore, we choose `epoch` = 20.

5) Impact of the (Packet Sending) `speed`: Its unit is packet per second (PPS). It can be obtained by calculating `speed=1/delay`, where `delay` represents the time interval between two consecutive probe packets sent by the host

(unit is μs). As shown in Fig. 12, clock drift with `speed` = 285.7 shows the earliest and most stable convergence. However, when the `speed` = 285.7, the corresponding CPU usage is also the largest (no matter in the system state or the user state), as shown in Fig. 13. In fact, there is an upper limit to the `speed`. If the `speed` becomes faster, the number of timestamps in each time slice will be higher. At this time, the offset will be calculated faster, causing the system to converge and stabilize earlier. However, too fast `speed` also means that a large number of timestamps are collected in a short time (larger `win_len`), which in turn will put a heavy burden on the CPU and cause drift/offset quality degradation. Therefore, after considering the CPU usage and synchronization accuracy, we choose `speed` = 71.4 (i.e., `delay` = 14 000 μs here) with the smallest CPU usage.
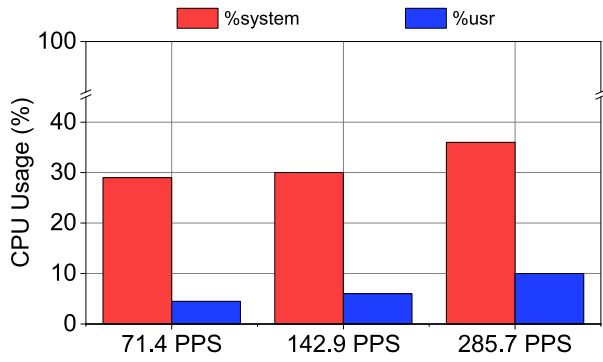
Fig. 13. Impact of the CPU load rate when speed varies, where speed = 71.4 has the minimum CPU usage.
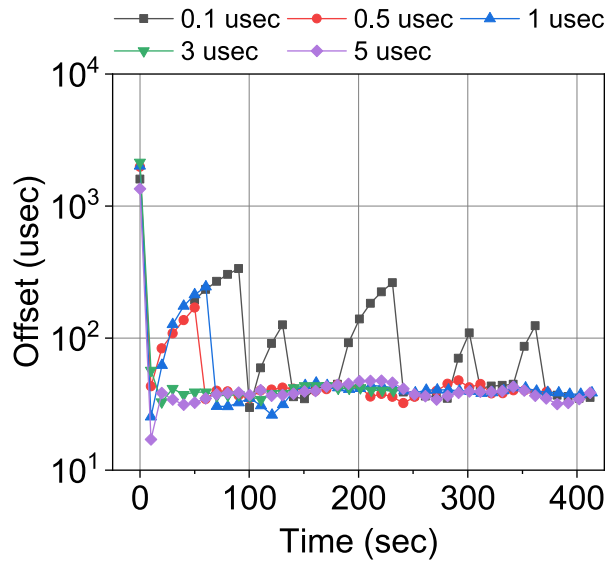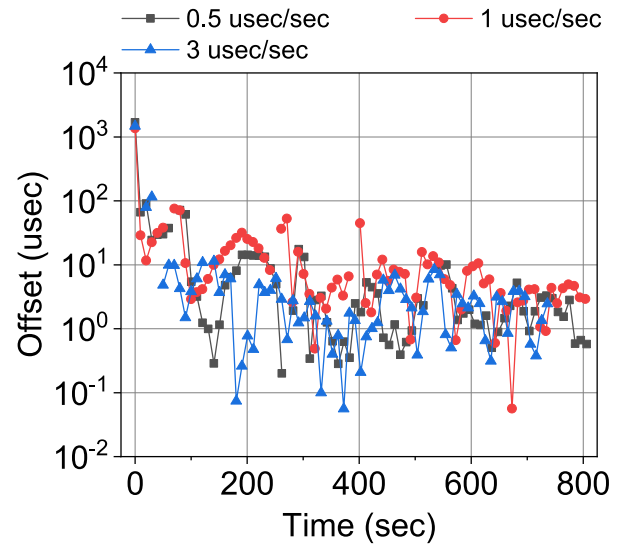

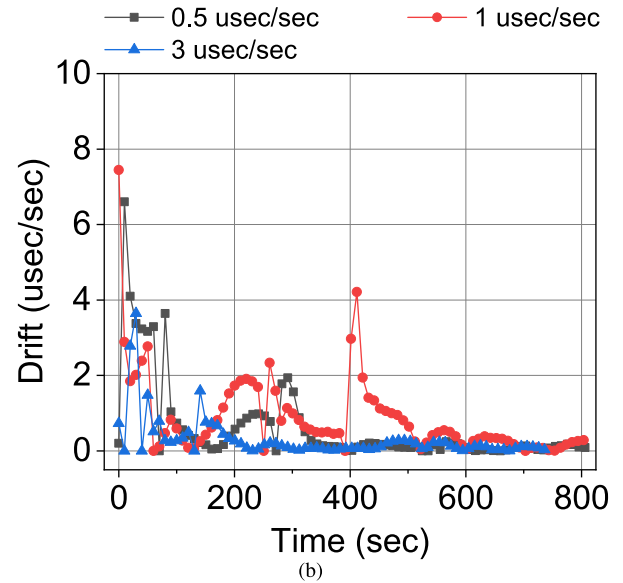Fig. 14. Impact of $\mathcal{T}_O$, where $\mathcal{T}_O = 3$ is the earliest stable.

Fig. 15. Impact of $\mathcal{T}_D$, where $\mathcal{T}_D = 3$ is the earliest stable. (a) Offset. (b) Drift.

Analysis of 3)–5): Our requirement for **SSA** is not only to ensure drift/offset stability, but also to ensure that a result can be output in a short interval to adjust the clock, which is also one of the ideal conditions for clock synchronization. In fact, if the packet sending speed is too fast, it will cause instability. This is the essential reason for using sliding windows.

6) Impact of the Offset Threshold $\mathcal{T}_O$: It is an important component of clock adaptive correction, as shown in Section III-D, with the unit being μs. In this experiment, we fix the drift threshold $\mathcal{T}_D = 0$, i.e., not adjusting $\mathcal{T}_D$. Then we use five different $\mathcal{T}_O$ values ($\mathcal{T}_O = 0.1, 0.5, 1, 3, 5$) to observe the effect on clock offset. As shown in Fig. 14, when $\mathcal{T}_O = 0.1, 0.5, 1$, the offset is very unstable and jitter is obvious; when $\mathcal{T}_O = 3$, the offset is the most stable. Therefore, we choose $\mathcal{T}_O = 3$. In fact, the difference of $\mathcal{T}_O$ affects the speed of convergence, but these offsets finally converge to around 40 μs.

7) Impact of the Drift Threshold $\mathcal{T}_D$: It is also an important component of clock adaptive correction, as shown in Section III-D, with the unit being μs/s. In this experiment, we fix the offset threshold $\mathcal{T}_O = 3$ according to the above $\mathcal{T}_O$

experiment, and use three different $\mathcal{T}_D$ values ($\mathcal{T}_D = 0.5, 1, 3$) to observe the effect on both clock offset and drift. As shown in Fig. 15(b), when $\mathcal{T}_D = 3$, clock drift is the most stable. Correspondingly, as shown in Fig. 15(a), when $\mathcal{T}_D = 3$, the clock offset is obviously the smallest on average. Therefore, we choose $\mathcal{T}_D = 3$. We find that the values of these offsets are significantly smaller than those in Fig. 14. This is because the frequency has been adjusted [Fig. 7 (left)], i.e., when the drifts are 0 in Fig. 15(b). The reason for frequency adjustments is based on $\mathcal{T}_D$, to make the stabilized offset more accurate.

Analysis of 6) and 7): According to Fig. 7, the threshold is variable. We do not expect the frequency to be adjusted all the time, which leads to instability of the clock. Therefore, during the entire adaptive correction process, the drift adjustment is getting slower, but the offset has been constantly adjusted to avoid the accumulation of offset errors. In short, adaptive correction allows the clock frequency and offset to be quickly
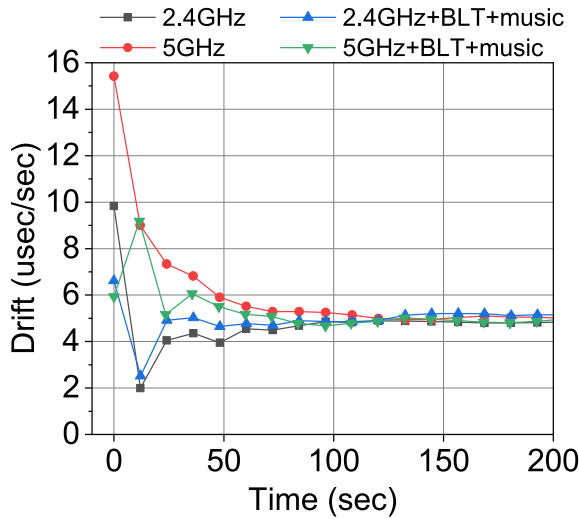
Fig. 16. Impact of the connection status, where the working environment with Bluetooth connected and playing music at the same time is stable earlier.
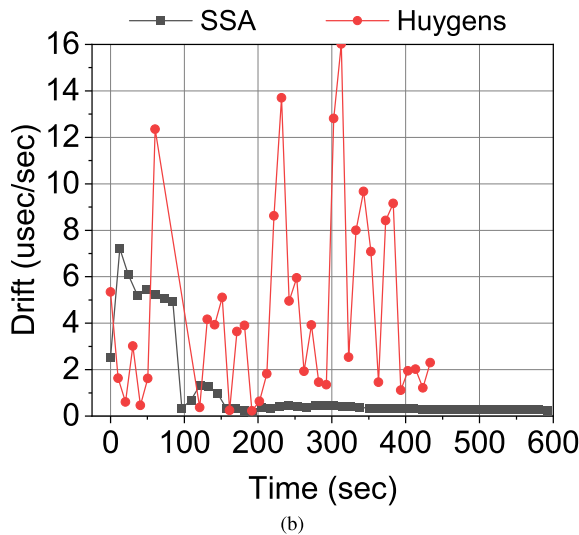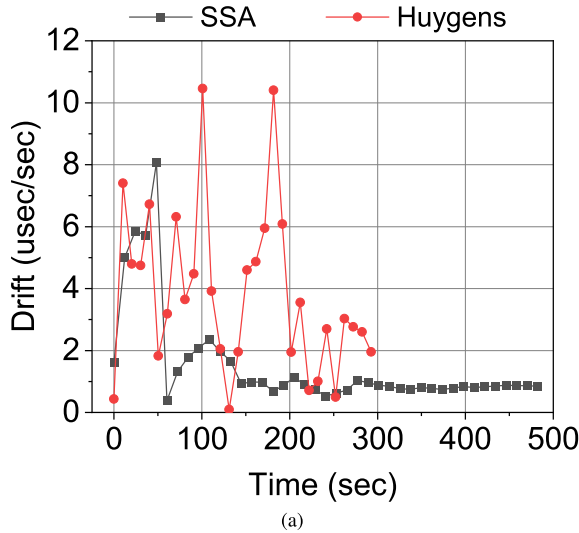

(a)

(b)

Fig. 17. In the actual working environment, the clock drift of **SSA** is obviously lower than that of the Huygens. (a) 2.4 GHz + BLT + music. (b) 5 GHz + BLT + music.

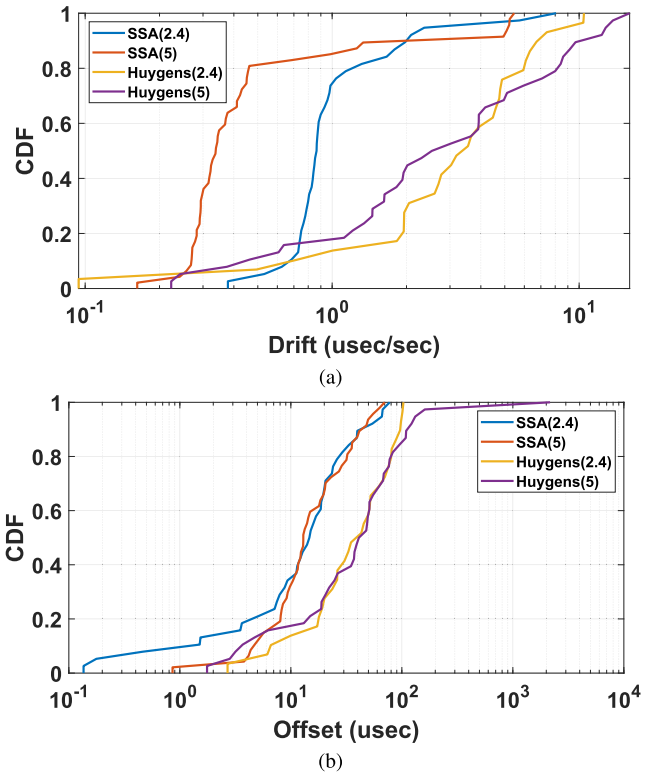adjusted after the system is started, so that the error can be quickly reduced to a small value.

(a)

(b)

Fig. 18. In the actual working environment, the CDF of **SSA** is obviously closer to the y-axis and performs better than that of the Huygens. (a) Drift. (b) Offset.

8) Impact of the Connection Status of the Smart Speaker: Our default smart speaker is working under 2.4/5 GHz WiFi. In this experiment, we observe the impact of drift connecting to Bluetooth (BLT) and playing music under default conditions. As shown in Fig. 16, we find that clock drift is stable earlier than the default setting when connecting to BLT and playing music, i.e., it seems to work better in a real working environment.

Discussion of Convergence: Parameters 1)–8) are different in sensitivity to convergence/stable time. Specifically, we can roughly sort their sensitivity according to the time values of each parameter in Figs. 8–16 when they all converge under different values as follows: 7) $\approx$ 6) $\gg$ 2) $>$ 4) $>$ 5) $>$ 3) $\approx$ 8) $>$ 1). 6) and 7) are the most sensitive because they are the key parameters that determine the adaptive correction (Section III-D) of the clock. If they are not tuned properly, the clock will be adjusted for a long time. Similarly, 2) and 4), which are second only to 6) and 7) in sensitivity, are also directly linked to our sliding window and smoothing techniques (Sections III-B and III-C). In general, the tuning of convergence-sensitive parameters needs to be handled with caution.

C. Results of System-Synthetic Experiments

We select the optimal parameters for system-synthetic experiments based on the results of the above-mentioned parameter tuning. In this experiment, we show the final clock drift and clock offset under the full firepower of **SSA** and
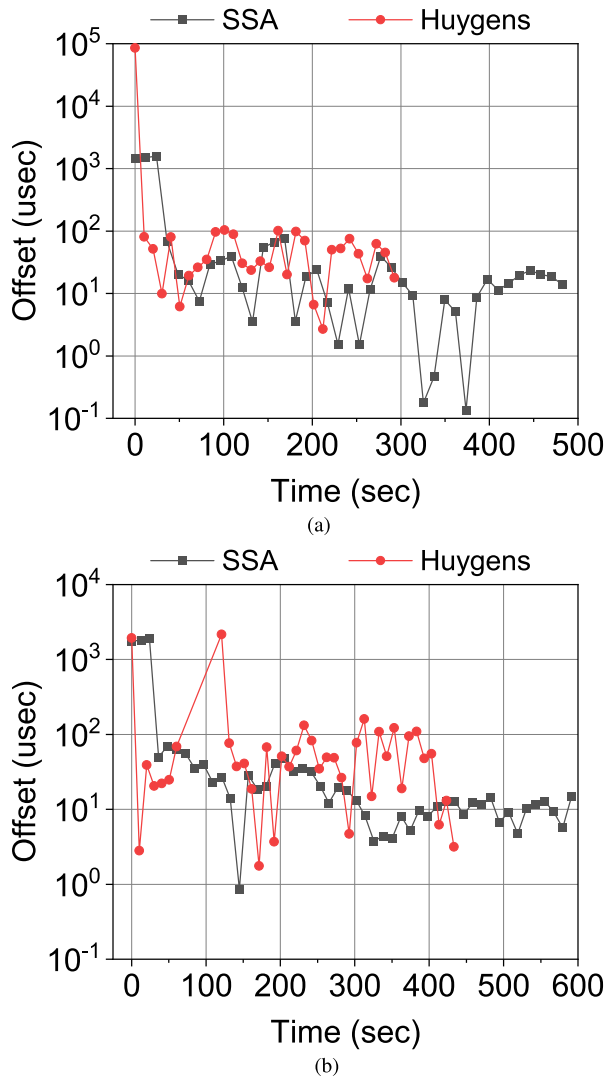
Fig. 19. In the actual working environment, the clock offset of **SSA** is basically lower than that of the Huygens. (a) 2.4 GHz + BLT + music. (b) 5 GHz + BLT + music.

observe how much our proposed **SSA** improves the Huygens clock synchronization scheme.[3]

1) Results of Clock Drift: As shown in Fig. 17, the experimental results show that the clock offset of **SSA** has obvious advantages compared with Huygens: the offset of **SSA** has already stabilized at a small value, but the offset of Huygens has not been stable and the value is always relatively large. For the `2.4 GHz + BLT + music` scenario, the drift of **SSA** stabilizes at an average of 0.835 μs/s after about 144 s of system startup. The average of all its drift is 1.30 μs/s, which is 3.03 times better than Huygens. For the `5 GHz + BLT + music` scenario, the drift of **SSA** stabilizes at an average of 0.331 μs/s after about 156 s of system startup. The average of all its drift is 0.91 μs/s, which is 4.90 times better than Huygens. For the comparison of different scenarios of **SSA**, the drift of `2.4 GHz + BLT + music` stabilizes earlier than that of `5 GHz + BLT + music`, but the value

---

[3]For a fair comparison, the first or first two or three consecutive large offset and corresponding drift values of **SSA** and Huygens when the system is just started are not included in the calculation of the average and CDF, because the offset values at these moments may be previous accumulation.

of `5 GHz + BLT + music` is significantly smaller when it stabilizes. In Fig. 18(a), we find the drift distribution of **SSA** is significantly closer to the y-axis, which also proves **SSA** has better clock drift.

2) Results of Clock Offset: As shown in Fig. 19, the experimental results show that the clock offset of **SSA** is still higher than that of Huygens as a whole. For the `2.4 GHz + BLT + music` scenario, the average of all its offset is 19.96 μs, which is 2.37 times better than Huygens. For the `5 GHz + BLT + music` scenario, the average of all its offset is 19.97 μs, which is 5.27 times better than Huygens. For the comparison of different scenarios of **SSA**, the offset variability trends of `2.4 GHz + BLT + music` and `5 GHz + BLT + music` are similar, but `2.4 GHz + BLT + music` fluctuates more obviously. What they have in common is that their offset values fluctuate slightly at about 10 μs in the end, and the best values are 0.135 μs (135 ns) and 0.859 μs (859 ns), respectively. Fig. 18(b) shows that the offset distribution of **SSA** is also closer to the y-axis, proving that **SSA** has better synchronization accuracy than Huygens.

V. CONCLUSION

Clock synchronization is fundamental, and its granularity is important. Among the clock synchronization solutions, the data-driven algorithms have attracted widespread attention in improving synchronization accuracy, and the state-of-the-art is the Huygens algorithm. However, directly adapting the Huygens algorithm to IoT devices is challenging. To address the problem, we propose an accurate microsecond-level clock synchronization, namely **SSA**, for IoT devices. We conduct extensive experiments with the Huawei Sound X in the actual WiFi-connected working environment. Experimental results show that **SSA** can achieve synchronization accuracy of around 20 μs.

REFERENCES

[1] Y. Wu, X.-Y. Li, Y. Liu, and W. Lou, "Energy-efficient wake-up scheduling for data collection and aggregation," *IEEE Trans. Parallel Distrib. Syst.*, vol. 21, no. 2, pp. 275–287, Feb. 2010.

[2] K. Fan, S. Sun, Z. Yan, Q. Pan, H. Li, and Y. Yang, "A blockchain-based clock synchronization scheme in IoT," *Future Gener. Comput. Syst.*, vol. 101, pp. 524–533, Dec. 2019.

[3] Y. Liu, J. Li, and M. Guizani, "Lightweight secure global time synchronization for wireless sensor networks," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, Apr. 2012, pp. 2312–2317.

[4] L. Beltramelli, A. Mahmood, P. Österberg, M. Gidlund, P. Ferrari, and E. Sisinni, "Energy efficiency of slotted LoRaWAN communication with out-of-band synchronization," *IEEE Trans. Instrum. Meas.*, vol. 70, pp. 1–11, 2021.

[5] P. Yadav, J. A. McCann, and T. Pereira, "Self-synchronization in duty-cycled Internet of Things (IoT) applications," *IEEE Internet Things J.*, vol. 4, no. 6, pp. 2058–2069, Dec. 2017.

[6] Y.-H. Wei, Q. Leng, S. Han, A. K. Mok, W. Zhang, and M. Tomizuka, "RT-WiFi: Real-time high-speed communication protocol for wireless cyber-physical control applications," in *Proc. IEEE 34th Real-Time Syst. Symp.*, Dec. 2013, pp. 140–149.

[7] S. Viswanathan, R. Tan, and D. K. Y. Yau, "Exploiting power grid for accurate and secure clock synchronization in industrial IoT," in *Proc. IEEE Real-Time Syst. Symp. (RTSS)*, Dec. 2016, pp. 146–156.

[8] P. Ferrari, A. Flammini, E. Sisinni, S. Rinaldi, D. Brandão, and M. S. Rocha, "Delay estimation of industrial IoT applications based on messaging protocols," *IEEE Trans. Instrum. Meas.*, vol. 67, no. 9, pp. 2188–2199, Sep. 2018.

[9] B.-Y. Ooi and S. Shirmohammadi, "The potential of IoT for instrumentation and measurement," *IEEE Instrum. Meas. Mag.*, vol. 23, no. 3, pp. 21–26, May 2020.

[10] A. Araujo et al., "Wireless measurement system for structural health monitoring with high time-synchronization accuracy," *IEEE Trans. Instrum. Meas.*, vol. 61, no. 3, pp. 801–810, Mar. 2012.

[11] T. Lindh, J. Wåhslén, I. Orhan, and D. Sturm, "Performance evaluation of time synchronization and clock drift compensation in wireless personal area networks," in *Proc. 7th Int. Conf. Body Area Netw.*, 2012, pp. 153–158.

[12] V. Jagadeeswari, V. Subramaniyaswamy, R. Logesh, and V. Vijayakumar, "A study on medical Internet of Things and big data in personalized healthcare system," *Health Inf. Sci. Syst.*, vol. 6, no. 1, pp. 1–20, Dec. 2018.

[13] A. De Angelis, P. Carbone, E. Sisinni, and A. Flammini, "Performance assessment of chirp-based time dissemination and data communications in inductively coupled links," *IEEE Trans. Instrum. Meas.*, vol. 66, no. 9, pp. 2474–2482, Sep. 2017.

[14] Z. Yuan, T. Bi, G. M. Muntean, and G. Ghinea, "Perceived synchronization of mulsemedia services," *IEEE Trans. Multimedia*, vol. 17, no. 7, pp. 957–966, Jul. 2015.

[15] D. T. Kemp, "Stimulated acoustic emissions from within the human auditory system," *J. Acoust. Soc. Amer.*, vol. 64, no. 5, pp. 1386–1391, 1978.

[16] R. C. Waters, "Time synchronization in spline," MERL, Cambridge, MA, USA, Tech. Rep. TR96-09, 1996.

[17] T. Picton, "Hearing in time: Evoked potential studies of temporal processing," *Ear Hearing*, vol. 34, no. 4, pp. 385–401, 2013.

[18] D. L. Mills, "Internet time synchronization: The network time protocol," *IEEE Trans. Commun.*, vol. 39, no. 10, pp. 1482–1493, Oct. 1991.

[19] J. C. Eidson, M. Fischer, and J. White, "Standard for a precision clock synchronization protocol for networked measurement and control systems," in *Proc. 34th Annu. Precis Time Time Interval Syst. Appl. Meet.*, 2002, pp. 243–254.

[20] J. Elson, L. Girod, and D. Estrin, "Fine-grained network time synchronization using reference broadcasts," *ACM SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, pp. 147–163, Dec. 2002.

[21] F. Sivrikaya and B. Yener, "Time synchronization in sensor networks: A survey," *IEEE Netw.*, vol. 18, no. 4, pp. 45–50, Jul. 2004.

[22] M. Maróti, B. Kusy, G. Simon, and Á. Lédeczi, "The flooding time synchronization protocol," in *Proc. 2nd Int. Conf. Embedded Networked Sensor Syst.*, Nov. 2004, pp. 39–49.

[23] H. Dai and R. Han, "TSync: A lightweight bidirectional time synchronization service for wireless sensor networks," *SIGMOBILE Mobile Comput. Commun. Rev.*, vol. 8, no. 1, pp. 125–139, Jan. 2004.

[24] K. S. Lee, H. Wang, V. Shrivastav, and H. Weatherspoon, "Globally synchronized time via datacenter networks," in *Proc. ACM SIGCOMM Conf.*, Aug. 2016, pp. 454–467.

[25] 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.

[26] M. Akhlaq and T. R. Sheltami, "RTSP: An accurate and energy-efficient protocol for clock synchronization in WSNs," *IEEE Trans. Instrum. Meas.*, vol. 62, no. 3, pp. 578–589, Mar. 2013.

[27] Y. Geng et al., "Exploiting a natural network effect for scalable, fine-grained clock synchronization," in *Proc. USENIX Symp. Networked Syst. Des. Implement.*, 2018, pp. 81–94.

[28] P. G. Kannan, R. Joshi, and M. C. Chan, "Precise time-synchronization in the data-plane using programmable switching ASICs," in *Proc. ACM Symp. SDN Res.*, Apr. 2019, pp. 8–20.

[29] K. He, C. An, J. H. Wang, T. Li, L. Zu, and F. Li, "Multi-hop precision time protocol: An internet applicable time synchronization scheme," in *Proc. IEEE/IFIP Netw. Operations Manage. Symp.*, Apr. 2022, pp. 1–9.

[30] 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.

[31] Y. Li et al., "Sundial: Fault-tolerant clock synchronization for datacenters," in *Proc. USENIX Symp. Oper. Syst. Des. Implement.*, 2020, pp. 1171–1186.

[32] A. Najafi and M. Wei, "Graham: Synchronizing clocks by leveraging local clock properties," in *Proc. USENIX Symp. Networked Syst. Des. Implement.*, 2022, pp. 453–466.

[33] Z. Yu, C. Jiang, Y. He, X. Zheng, and X. Guo, "Crocs: Cross-technology clock synchronization for WiFi and ZigBee," in *Proc. EWSN*, 2018, pp. 135–144.

[34] R. Hofmann, D. Grubmair, C. A. Boano, and K. Römer, "X-Sync: Cross-technology clock synchronization among off-the-shelf wireless IoT devices," in *Proc. IEEE 46th Conf. Local Comput. Netw. (LCN)*, Oct. 2021, pp. 115–122.

[35] M. Tortonesi et al., "Leveraging Internet of Things within the military network environment—Challenges and solutions," in *Proc. IEEE World Forum Internet Things (WF-IoT)*, Dec. 2016, pp. 111–116.

[36] S. K. Mani, R. Durairajan, P. Barford, and J. Sommers, "An architecture for IoT clock synchronization," in *Proc. 8th Int. Conf. Internet Things*, Oct. 2018, pp. 1–8.

[37] S. Wang, Y. Hou, F. Gao, and S. Ma, "A novel clock synchronization architecture for IoT access system," in *Proc. 2nd IEEE Int. Conf. Comput. Commun. (ICCC)*, Oct. 2016, pp. 1456–1459.

[38] D. Resner, A. A. Fröhlich, and L. F. Wanner, "Speculative precision time protocol: Submicrosecond clock synchronization for the IoT," in *Proc. IEEE 21st Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2016, pp. 1–8.

[39] C. Cortes and V. Vapnik, "Support-vector networks," *Mach. Learn.*, vol. 20, no. 3, pp. 273–297, Jan. 1995.

[40] Q. Wu and D. Zhou, "SVM soft margin classifiers: Linear programming versus quadratic programming," *Neural Comput.*, vol. 17, no. 5, pp. 1160–1187, May 2005.

[41] L. Fanucci et al., "Sensing devices and sensor signal processing for remote monitoring of vital signs in CHF patients," *IEEE Trans. Instrum. Meas.*, vol. 62, no. 3, pp. 553–569, Mar. 2012.

[42] G. Mois, S. Folea, and T. Sanislav, "Analysis of three IoT-based wireless sensors for environmental monitoring," *IEEE Trans. Instrum. Meas.*, vol. 66, no. 8, pp. 2056–2064, Aug. 2017.

[43] C. Lenzen, P. Sommer, and R. Wattenhofer, "PulseSync: An efficient and scalable clock synchronization protocol," *IEEE/ACM Trans. Netw.*, vol. 23, no. 3, pp. 717–727, Jun. 2014.

[44] H. Kim, X. Ma, and B. R. Hamilton, "Tracking low-precision clocks with time-varying drifts using Kalman filtering," *IEEE/ACM Trans. Netw.*, vol. 20, no. 1, pp. 257–270, Feb. 2011.

[45] X. Yan and X. G. Su, *Linear Regression Analysis: Linear Regression Analysis: Theory and Computing*. Singapore: World Scientific, 2009.

[46] (2023). *Huawei Sound X*. [Online]. Available: https://consumer.huawei.com/en/speakers/sound-x/

Zhuochen Fan received the Ph.D. degree in computer science from Peking University, Beijing, China, in 2023, under the supervision of Tong Yang.

He has published papers the IEEE Real-Time Systems Symposium (RTSS), IEEE International Conference on Data Engineering (ICDE), IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, the International Conference on Parallel Processing (ICPP), the IEEE International Conference on Network Protocols (ICNP), etc. His research interests include network measurements, data stream processing and algorithms, and clock synchronization.

Yanwei Xu received the Ph.D. degree in computer science from Tongji University, Shanghai, China, in 2012.

He is currently a Senior Researcher with the Theory Laboratory, Huawei Hong Kong Research Center, Hong Kong. His main research interests include lie in software defined networks, network measurements, and clock synchronization.
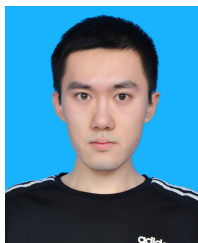
Peng Liu received the B.S. and M.S. degrees in computer science from Peking University, Beijing, China, in 2018 and 2021, respectively.

He has participated several articles in network area, advised by Tong Yang. His research interests include networks and data stream processing.

Xiaodong Li received the B.E. degree in the internet of things (IoT) engineering from the University of Science and Technology, Beijing, China, in 2019. He is currently pursuing the M.S. degree in computer science with the Peking University Shenzhen Graduate School, Shenzhen, China.

His research interests include mainly focus on data stream processing and programmable switches.

Ruwen Zhang received the B.S. degree in mathematics from Peking University, Beijing, China, in 2021, where he is currently pursuing the master's degree, under the supervision of Tong Yang.

He has participated several articles in network area. His research interests include networks and data stream processing.

Tong Yang (Member, IEEE) received the Ph.D. degree in computer science from Tsinghua University, Beijing, China, in 2013.

He has visited the Institute of Computing Technology, Chinese Academy of Sciences (CAS), Beijing. He is currently an Associate Professor with the School of Computer Science, Peking University, Beijing. He has published papers in The Conference of the ACM Special Interest Group on Data Communication (SIGCOMM), The ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD), The Conference of the ACM Special Interest Group on Management of Data (SIGMOD), The USENIX Symposium on Networked Systems Design and Implementation (NSDI), USENIX ATC, ICDE, VLDB, INFOCOM, ICNP, IEEE/ACM Transactions on Networking, IEEE Transactions on Computers, and IEEE Journal on Selected Areas in Communications. His research interests include network measurements, sketches, IP lookups, Bloom filters, and KV stores.

Dr. Yang is currently an Associate Editor of *Knowledge and Information Systems*.

Wenfei Wu (Member, IEEE) received the Ph.D. degree in computer science from the University of Wisconsin–Madison, Madison, WI, USA, in 2015.

He was an Assistant Professor with the Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing, China. He is currently an Assistant Professor with the School of Computer Science, Peking University, Beijing. He has published papers in The Conference of the ACM Special Interest Group on Data Communication (SIGCOMM), The ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD), The USENIX Symposium on Networked Systems Design and Implementation (NSDI), The ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), The IEEE International Conference on Computer Communications (INFOCOM), The ACM International Conference on Emerging Networking Experiments and Technologies (CoNEXT), The IEEE International Conference on Distributed Computing Systems (ICDCS). IEEE/ACM Transactions on Networking, and IEEE Transactions on Computers. His research interests include the information infrastructure for distributed applications, such as machine learning, cloud computing, and big data.

Dr. Wu has served as the TPC Co-Chair for ICNP 2022 and 2021, IM 2021, and SIGCOMM 2020 and 2019 Poster and Demo.

Yuqing Li (Associate Member, IEEE) received the B.S. degree in communication engineering from Xidian University, Xi'an, China, in 2014, and the Ph.D. degree in electronic engineering from Shanghai Jiao Tong University, Shanghai, China, in 2019.

From 2019 to 2020, she was a Post-Doctoral Fellow with The Hong Kong University of Science and Technology, Hong Kong. From 2020 to 2022, she was a Researcher with the Huawei Hong Kong Research Center, Hong Kong. She is currently an Associate Professor with the School of Cyber Science and Engineering, Wuhan University, Wuhan, China. Her current research interests include cloud/edge computing, network measurement, distributed machine learning, data privacy and security, and algorithmic network economics.

Li Chen received the B.E., M.Phil., and Ph.D. degrees from The Hong Kong University of Science and Technology (HKUST), Hong Kong, in 2011, 2013, and 2018, respectively.

He is currently a Researcher with the Zhongguancun Laboratory, Beijing, China. His research interests include data center networks, distributed systems, parallel computing, AI and machine learning, and OAM.

Gong Zhang (Member, IEEE) is a Chief Researcher Scientist in future network architecture of Huawei 2012 Labs. He was a Senior Researcher, leading future internet research and cooperative communication, and did Mobility research program since 2005. In 2009, he is a Principal Researcher in charge of advance network technology research department, leading researches of future network, distributed computing, Database system and data analysis. He proposed stream-based research system in network to maintain network and find added value for carries. He was a Principal Researcher, leading system group in data mining and machine learning since 2012. He is over 18 years Research experience of system architect in Network, distributed system and communication system. He had contributed more than 90 patents globally. His major research directions are network architecture and large-scale distributed system.