electronics

Article

Single-Timestamp Skew Correction (STSC) in V2X Networks

Muhammad Usman Hashmi [1], Muntazir Hussain [2,*], Muhammad Babar [3] and Basit Qureshi [4]

[1] Department of Computer Science, Bahria University, Islamabad 44230, Pakistan
[2] Department of Electrical and Computer Engineering, Air University, Islamabad 44230, Pakistan
[3] Robotics and Internet of Things Lab, Prince Sultan University (PSU), Riyadh 66833, Saudi Arabia
[4] College of Computer and Information Sciences, Prince Sultan University (PSU), Riyadh 66833, Saudi Arabia
* Correspondence: muntazir_hussain14@yahoo.com or muntazir.hussain@mail.au.edu.pk

Abstract: Modern vehicles nowadays have many capabilities apart from the basic function of driving. These are now intelligent, smart, and can communicate over the Internet. A vehicle-to-everything (V2X) wireless network represents a network where vehicles communicate vital sensor data with other vehicles, pedestrians, and fixed infrastructure over the internet. There are various challenges in V2X communication that may affect the efficiency of autonomous devices, systems, and infrastructure. Of the many challenges, time synchronization among many devices in V2X networks is a key challenge. In a V2X network, all nodes within the network need to be time-synchronized; this is essential for task scheduling, computation off-loading, event sequencing, resource sharing, and efficient utilization of resources in the network. In recent works, many researchers have addressed time synchronization in V2X networks by considering multiple timestamps in order to estimate the time skew offset with varying results. In this paper, we consider a skew-based approach, namely, a single-timestamp skew correction (STSC) for time synchronization in V2X networks. The proposed method needs a single timestamp to estimate time skew at the hardware level with the help of physical-layer time synchronization using symbol timing recovery. Implementation results prove that the STSC accurately synchronizes the nodes in phase and frequency, therefore resulting in a greater accuracy and better energy savings in the V2X networks.

Keywords: skew correction; time synchronization; vehicle-to-everything wireless systems

Citation: Hashmi, M.U.; Hussain, M.; Babar, M.; Qureshi, B. Single-Timestamp Skew Correction (STSC) in V2X Networks. *Electronics* **2023**, *12*, 1276. https://doi.org/10.3390/electronics12061276

Academic Editors: Norman Franchi and Mohamed Khalaf-Allah

Received: 30 January 2023
Revised: 2 March 2023
Accepted: 4 March 2023
Published: 7 March 2023

1. Introduction

In current times, one of the enabling technologies for smart vehicles on the roads is the vehicle-to-everything (V2X) network. In a V2X, smart vehicles can communicate with other vehicles, devices, and infrastructure within a network. This large network encompasses vehicle-to-pedestrian (V2P), vehicle-to-infrastructure (V2I), and vehicle-to-cloud (V2C) communications [1,2]. The motivation behind V2X networks is to improve road safety, reduce congestion, and enhance the overall driving experience. Some of the key benefits of V2X technology include improved safety, reduced congestion, enhanced driving experience, collision detection and avoidance, traffic management and control, sensor data sharing, emergency evacuation control, and infotainment [3–5]. V2X technology is indeed a technology that enables a cost-effective, safe, and secure environment for urban smart transportation. Some real-life application scenarios of V2X technology include intersection safety, emergency vehicle prioritization, autonomous driving, smart parking, and road weather monitoring.

In a V2X network, the communication can be from a vehicle to an infrastructure, pedestrian, or another vehicle. In each network, communication requirements are different because of their mobility and applications. For instance, in a V2I network, vehicles are mobile and the infrastructure is stationary and fixed. Similarly, in a V2P network, vehicles have a high mobility and pedestrians have low mobility, and in a V2V network, both communicating parties are vehicles with a high mobility. Other than mobility, applications are of

versatile nature in each case and that makes the requirements versatile. Thus, in a V2X network the applications and their requirements are very much different from each other [6,7]. Some applications need a high throughput and low latency whereas in some applications, throughput does not matter. In some applications, data reliability is the only demand, whereas in some applications, just sending data without confirmation is enough [8,9]. Such a versatile nature of applications comes with lots of challenging tasks, and one the major issue of time synchronization among the nodes of the V2X network. As the network is distributed, there are many clocks within the network. No matter how precise the clocks are, they still drift away after some time and require synchronization [10,11]. An interesting survey on the latest clock synchronization techniques based on statistical signal processing methods in wireless sensor networks (WSNs) was provided in [12]. The authors concluded the clock synchronization performance could be evaluated by employing statistical signal processing approaches. The clock synchronization in a loosely coupled distributed network and the efficient clock synchronization unit implementation of a fault-tolerant global time base was presented in [13]. However, it is important to note that these delays are at the application layer and will not affect the extraction of a time offset from physical layer. Most of the nodes in a V2X network are mobile, and this requires special synchronization techniques. Most of the V2X applications are time-sensitive or much less time-offset-tolerant; for example, event notification might not be helpful if it did not get received at the time of event. An exact sequence of events cannot be found without a synchronized clock among the whole network. Out-of-order events might not be able to give a full picture of what is happening. To address that, we have time synchronization techniques and protocols.

To the best of our knowledge, most of the time synchronization techniques available in the literature depend on the exchange of multiple timestamps for the estimation of the time skew. One timestamp only compensates for the time phase offset. To know the frequency difference of the clock, multiple timestamps are analysed [14]. The method proposed in this paper needs only one timestamp to estimate the time skew at the hardware level, with the help of a physical-layer time synchronization using symbol timing recovery. This skew is derived from a hardware clock and hence can be used to correct the software clock skew as the software clock is also running on the same hardware clock. Therefore, in the proposed single-timestamp skew-correction (STSC) method, only one timestamp is needed. If a high level of skew estimation efficiency is required, then multiple timestamp packets can be used with the STSC method.

2. Related Work

V2X nodes consist mostly of vehicles and pedestrians, which creates decentralized and distributed network, and each of the node is connected using wireless media. Hence, it is highly required to have the same clock among all the nodes of the network. Without a single notion of time, time-sensitive services offered by a V2X network will not function [15,16]. For the implementation of a single clock, synchronization protocols and techniques play a vital role. These protocols adjust the phase and skew offset of the clocks in the network. In order to fully understand the working of the protocols, clocks and their imperfections need to be discussed.

Every electronic device that has the capability to compute must keep track of time in order to process instructions in a sequential manner and also to keep events in order. This time-keeping capability is provided by a clock circuit. In reality, these clocks are actually timers. The heart of the clock circuit is a quartz crystal which is kept under tension. Under this condition, quartz crystals oscillate, and the frequency of the oscillations produced depends on the type of crystal used and the provided tension. These oscillations become the input to counter registers and holding registers. These registers actually store integer values. A fixed value is stored in a holding register, which is used as a reference to count the oscillations before one clock tick. This value is first loaded in a counter register and then on every oscillation of the quartz crystal, the counter register decrements by one. When this counter register reaches zero, a clock tick is generated, and the counter

register is reloaded from the holding register. This clock tick is actually a timer interrupt and used by digital processors and every other component of the computing device [10,14]. Adjusting the crystal's frequency and value of the holding register controls the frequency of the generated clock ticks. The clock tick's frequency is always less than the frequency of the crystal.

These clock ticks can be related to actual time in one of the following two methods. The first method has a specific date and time already stored in memory and whenever such a system gets turned on, it requires the current date and time which can be taken from a user or any other source. The current date and time is converted in ticks from the start date that was already stored in memory and then stored in counting registers to continue from there. The second method is widely in use. Most of the computing devices have a separate battery which keeps the clock running even when the device is powered off [17]. In this way, devices can continue to boot up without asking any date time information. Every clock is derived from this hardware clock running at different clock tick counts. These derived clocks are known as software clocks. A software clock can be used to provide the actual time or just a sequence of events. When the device is centrally controlled by one clock and all the processes are running on the same machine then it does not matter if the clock is not perfectly synchronized. All the processes will still relate in time with each other. However, in the case of a distributed system, where each device is running its own clock, then synchronization among the clock becomes critical.

These digital clocks of distributed systems are based on a superfine quartz crystal, but the oscillations produced may still drift a bit from each other and hence may go totally out of synchronization over a longer period. It is not possible to run all the clocks at precisely the same frequency. Due to this slight imperfection, software clocks gradually drifts away from each other and this imperfection is known as the clock skew [14,18]. This synchronization error causes time-sensitive applications to become ineffective because in such applications, time is usually associated with events and a sequencing of events. In any distributed system, having a synchronized clock is desirable throughout the network and synchronization protocols are used for that purpose. Some of the important protocols used at the application layer of V2X systems are discussed below.

One of the synchronization protocols is reference broadcast synchronization (RBS). In RBS, a master node has the capability to instruct nodes and it broadcasts a beacon to every other node in the network. On the reception of the beacon/instruction from the master node, neighbouring nodes share time with each other and correct their own clocks [19]. Time is always shared in a packet known as timestamp which help the neighbouring nodes to synchronize. There are two major methods to correct the clock. Either each node keeps a record of the time offsets of other nodes and let the clocks run untethered or the value of the timer can be updated accordingly. However, any abrupt change in the timer values can cause a missing time issue, and relating it with actual events becomes difficult. An easy approach is to either speed up or slow down a clock to gradually synchronize it in time phase. RBS uses multiple timestamp packets in order to estimate the time skew. A single timestamp exchange is not sufficient.

The timestamp synchronization (TSS) approach is also based on timestamp sharing. However, in TSS, unlike RBS, timestamps are not enclosed in separate packets. The timestamps are enclosed in control or data packets. Efficiency can be achieved in this way by reducing the number of packets exchanged [14]. However, multiple timestamps, which are sent in other packets, are still required in order to compensate for the clock skew.

Another approach is lightweight time synchronization (LTS), which creates a spanning tree of the network nodes following a centralized algorithm [14]. This algorithm creates pairs of nodes and synchronization is achieved in pairs. A master node has the perfect time and is considered as the root node of the tree having a time reference. LTS does not synchronize without demand.

The timing-sync protocol for sensor networks (TPSN) divides time synchronization in two phases, namely, a level-discovery phase and a synchronization phase [20,21]. First, all the nodes determine their level by transmitting messages to their neighbouring nodes in the level-discovery phase. Once all the nodes are kept in the multiple-level hierarchy, the synchronization phase starts. In this phase, each node share timestamps in a defined hierarchy and synchronizes with the master node.

Similarly, in the flooding time synchronization protocol (FTSP), a master node floods the timestamp to each node in the network. On receiving of timestamp, each node sets its clock accordingly. This method is one of the promising methods of time synchronization [22,23].

To the best of our knowledge, all of the above methods uses multiple timestamps to estimate the clock skew and use it for correcting the clock frequency offset [11,14]. These methods are unable to compute the skew with just one timestamp and hence, by using multiple timestamps, the energy efficiency gets compromised. It increases computations and the number of transmissions required. In the proposed STSC method in this paper, only one timestamp is enough to compute the clock skew and correct the clock in both phase and frequency. The main contribution in this paper are: (1) A time synchronization approach that extracts the clock skew using just one timestamp. (2) The energy efficiency is increased because multiple timestamps are no longer needed to extract the clock skew of the application layer's clock. (3) The STSC method is implemented to show that skew correction can be done using one timestamp.

The rest of the paper is structured as follows. In Section 2, a literature review of the current state-of-the-art techniques is mentioned. Section 3 proposes our methodology. Section 4 shows how the proposed methodology can be implemented. It also elaborates on the results obtained from the implementation. Section 5 concludes the research idea.

3. Proposed Methodology

Time synchronization is a process of synchronizing the phase as well as the frequency of a clock. One timestamp packet can synchronize the clock in phase, but multiple timestamps are used for the estimation of the frequency offset. Thus, in general, one timestamp is not enough to synchronize in both phase and frequency. The idea proposed here uses a single timestamp for the phase and frequency correction. It relies on the fact that the software clock is also derived from the hardware clock of the node. Therefore, if one can estimate the skew at the physical layer, which is that of the hardware clock, then it can be translated to the software clock for correction. This is because the software clock's skew is caused by imperfections of the hardware clock. The software clock's skew computation is the same whether two nodes share multiple timestamps to estimate the skew or whether the skew of the hardware clock is computed using one timestamp and then translated to the software clock. Keeping this in mind, symbol timing recovery at the physical layer can be used to estimate the skew offset of the hardware clock [24,25]. Every communication system at the physical layer requires symbol recovery, which can be done using multiple methods including a phase lock loop (PLL). Once the skew is estimated and corrected, the phase offset can be removed using the contents of the same timestamp packet. In this way, the STSC needs only one timestamp to synchronize both phase and skew.

The STSC approach is illustrated in Figure 1. In this approach, when a V2X node (e.g., a vehicle) wants to synchronize in time with another node such as a vehicle, pedestrian or infrastructure within a V2X wireless system, it sends one timestamp which includes the node's current time.

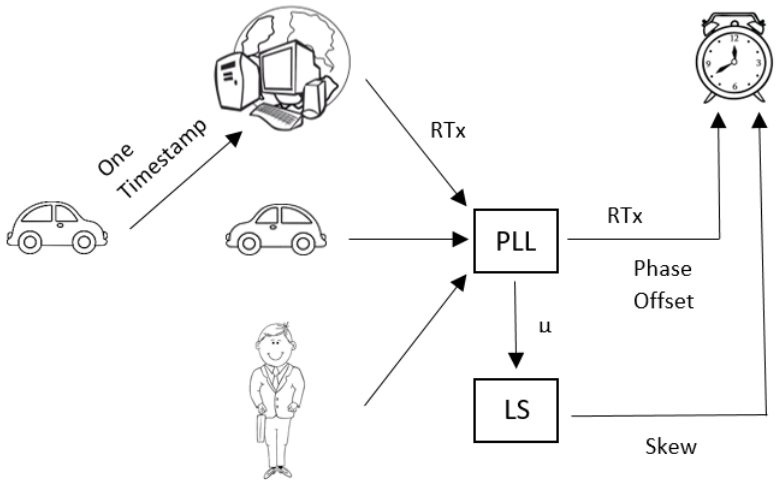

Figure 1. Single-timestamp skew-correction (STSC) approach.

For any wireless communication, symbol timing recovery is used at the receiver side in order to recover the received symbols. When the receiving node receives the timestamp packet at the physical layer, it passes the symbols to the PLL for symbol timing recovery and tries to track the error. The error signal provides the fractional offset μ, which relates to the clock skew. A least squares (LS) method is applied on μ for extracting the hardware clock skew. This estimated skew is now translated to the software clock for correction. Hence, the skew correction of the software clock is completed, and for the phase correction, it already has the timestamp. Thus, the STSC uses one timestamp to correct the phase and the skew of the receiving node's clock.

The whole process of STSC can also be explained with the help of flowchart shown in Figure 2. At the application layer of node A, a software clock is used to generate a timestamp. That timestamp representing the time of node A is packed into an STSC packet, which is eventually sent over the medium with the help of the physical layer. The physical layer deals with all the transmission details such as the modulation scheme, data rate, etc. This packet is transmitted to node B. At node B, the physical layer tries to recover the transmitted symbols which were affected during transmission. To recover the symbols, symbol timing recovery is conducted using multiple components. After passing the data from the matched filter, a timing error detector tries to extract the timing offset whereas a loop filter tries to track the error in real time [26,27]. On the computations of the loop filter, the interpolation control determines how to correct the upcoming symbols and instructs the interpolator accordingly. The output of the interpolation control specifically can adjust the time offset and is known as the fractional interval μ. Applying the least squares (LS) method to the output of the interpolation control leads to the skew estimation. This skew is used to correct and synchronize the time skew of node B's clock. The output of the interpolator, which consists of the received symbols, contains the clock value of node A and hence can be used to correct the time phase error of node B's clock. Once the time phase and skew are corrected, the clock is synchronized with the clock of node A.

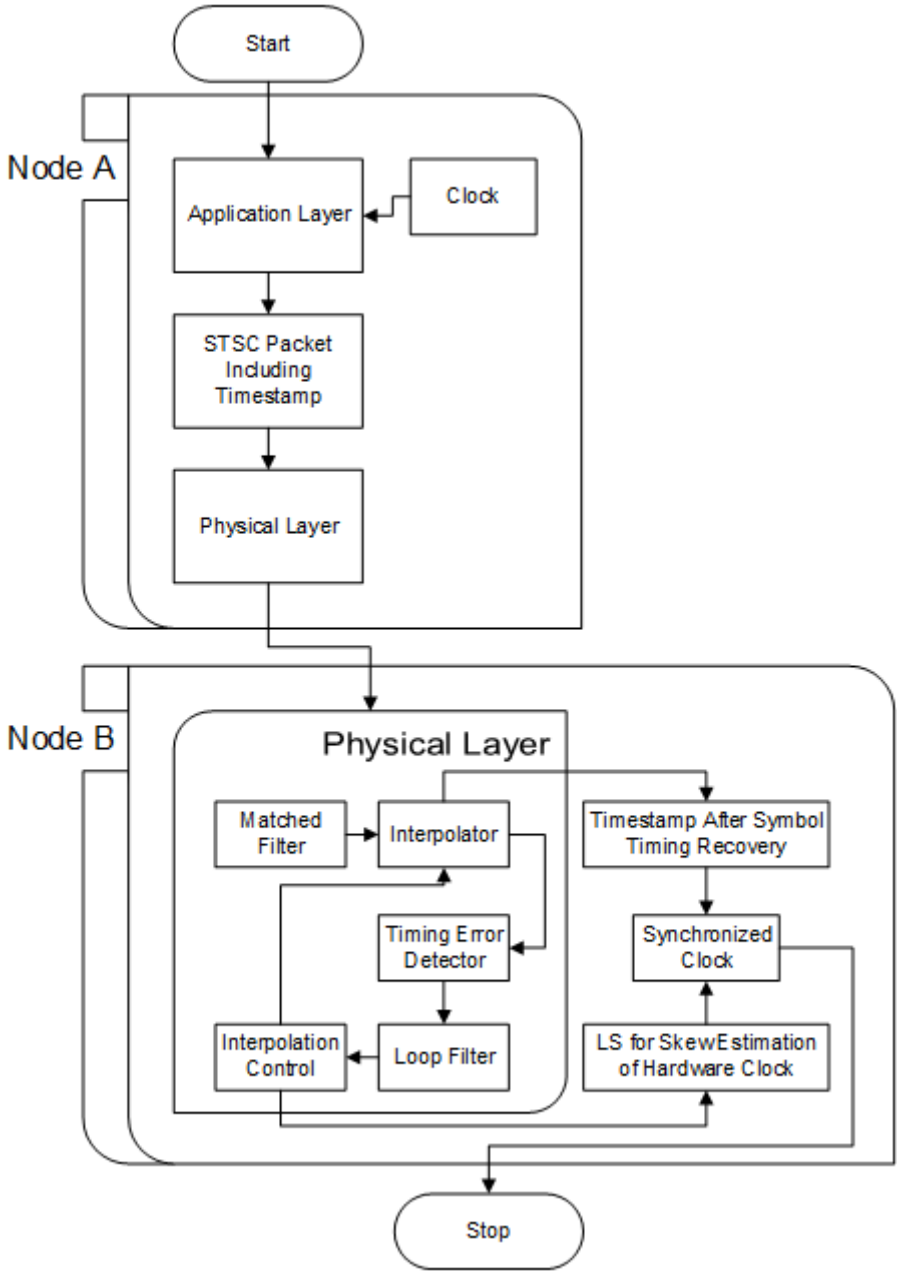

Figure 2. Flowchart of the single-timestamp skew-correction (STSC) approach.

4. Mathematical Analysis

Mathematically, the software clock can be defined as,

$$C(t) = k \int_{t_0}^{t} \omega(t)dt + C(t_0) \tag{1}$$

where t is the time and $\omega(t)$ is the angular frequency of the oscillator of the hardware clock [17,18,20]. If we simplify, then the software clock can be written as

$$S_C = st + \phi \tag{2}$$

where ϕ is the phase and s is the frequency of the software clock S_C. Using Equation (2), the software clocks S_C^A and S_C^B for two synchronizing nodes A and B can be written as

$$S_C^A = s^A t + \phi^A \tag{3}$$

$$S_C^B = s^B t + \phi^B \tag{4}$$

The stimestamp received from node A to node B have the phase ϕ^A included as the content of the message and hence, it is known at node B. Node B also knows its own phase ϕ^B because it simply reads the content from its own clock. The timing phase difference ϕ^o can now be computed as

$$\phi^o = \phi^B - \phi^A \tag{5}$$

or

$$\phi^B_{sync} = \phi^A + \phi^o \tag{6}$$

Once node B knows about the phase offset ϕ^o and ϕ^A, it adds these two quantities to synchronize its own clock. Hence, $\phi^B = \phi^B_{sync}$ which the synchronized phase of node B.

Now, for the computation of the skew offset, the output of the matched filter $x[nT]$ is fed to the interpolator to find the kth interpolant that is defined as

$$\begin{aligned} x((m(k)+\mu(k))T) = &\left(\frac{\mu(k)^3}{6} - \frac{\mu(k)}{6}\right)x((m(k)+2)T) - \left(\frac{\mu(k)^3}{2} - \frac{\mu(k)^2}{2} - \mu(k)\right) \\ &x((m(k)+1)T) + \left(\frac{\mu(k)^3}{2} - \mu(k)^2 - \frac{\mu(k)}{2} + 1\right)x(m(k)T) \\ &- \left(\frac{\mu(k)^3}{6} - \frac{\mu(k)^2}{2} + \frac{\mu(k)}{3}\right)x((m(k)-1)T) \end{aligned} \tag{7}$$

where μ is the fractional offset column vector of n elements and $\mu(k)$ represents the kth element of the vector. If we apply the least squares method [28], we can find the skew offset s^B_{sync} and c as

$$\begin{bmatrix} s^B_{sync} \\ c \end{bmatrix} = (\mu^T \mu)^{-1} \mu S \tag{8}$$

where S are the received samples and c is a constant. s^B_{sync} is the skew computed after achieving synchronization. The synchronized clock of node B can now be written with the help of Equations (4), (6), and (8) as

$$S^B_{Csync} = s^B_{sync} t + \phi^A + \phi^o \tag{9}$$

Hence, the fractional interval leads to the skew offset computation and can be applied to the computation of the software clock for synchronization.

The STSC method also optimizes the energy utilization of the V2X systems. By using one STSC packet, the energy required to synchronize is just the energy required to transmit a single timestamp. It can be mathematically written as

$$E_{STSC} = E_b N_b \tag{10}$$

where E_b is the energy per bit and N_b is the number of bits in one packet. However, if any other time synchronization protocol is used, then it requires multiple timestamp exchanges in order to compute the clock skew. The energy required for such protocols can be written as

$$E_{MT} = E_b N_b N_T \tag{11}$$

where E_{MT} is the energy required for multiple-timestamp protocols and N_T is number of timestamps exchanged to compute the skew.

Now, the efficiency η can be computed as

$$\eta = \frac{E_{MT}}{E_{STSC}} \tag{12}$$

By using Equations (10) and (11), we have

$$\eta = \frac{E_b N_b N_T}{E_b N_b} = N_T \tag{13}$$

Hence, theoretically this STSC approach increases the efficiency by N_T times. In this paper, we considered an efficient STSC for the time synchronization in V2X networks to estimate the time skew at the hardware level with the help of a physical-layer time synchronization using symbol timing recovery. In the future, the proposed STSC technique can be extended to the synchronization of a whole network of nodes as in [29,30] and to the synchronization of scenarios such as task offloading in mobile edge computing as in [31,32], which will be a good way forward for upcoming research articles.

5. Implementation and Results

The proposed idea was implemented on two floating-point DSP TMDSDSK6713 kits (DSK) manufactured by Texas Instruments. These kits provided low-cost development platforms and could be used for the implementation of physical-layer concepts such as symbol timing recovery. These kits acted like network nodes. There were two nodes in the implementation scenario, as shown in Figure 3.

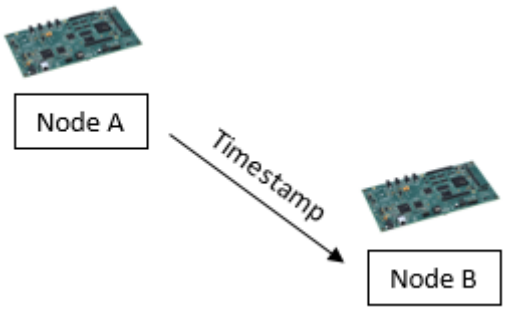

Figure 3. TMDSDSK6713 kits acting as nodes.

Hence, two DSP kits were used as node A and node B. The hardware clock of both kits had a known time skew of around 3 µs. However, this skew needed to be computed in wireless systems and was not already known. Node A had to synchronize in time with node B. Thus, node A sent a timestamp to node B. That timestamp included the sender's clock, which was used by the receiving node B to remove the clock phase offset. For the correction of the clock skew offset, node B extracted the time skew of the hardware clock using symbol timing recovery. Figure 4 shows the procedure adopted for receiving a timestamp at the physical layer, finding the skew, and applying it to the software clock of node B.

When two nodes try to synchronize at the physical layer, the received packet must undergo a symbol timing recovery [33,34]. One such system using PLL was used here. The symbols received in timestamp packets were binary PAM symbols. These symbols passed through the interpolator, timing error detector, loop filter, and interpolation control. A zero-crossing timing error detector (ZC-TED) was used to detect the timing error with the help of symbol values crossing at the zero level [35]. The loop filter tried to track the timing error detected by the ZC-TED. The interpolation control instructed the interpolator to change the sampling instant so that the next sample could be received with less error. Every next symbol error was detected, tracked, and the sampling instant kept changing until it completely tracked the error. These components were implemented for the computation of the fractional offset present between the sender and receiver clocks [33]. The output of the interpolation control was the fractional interval μ, which is a very important signal for extracting the time skew.

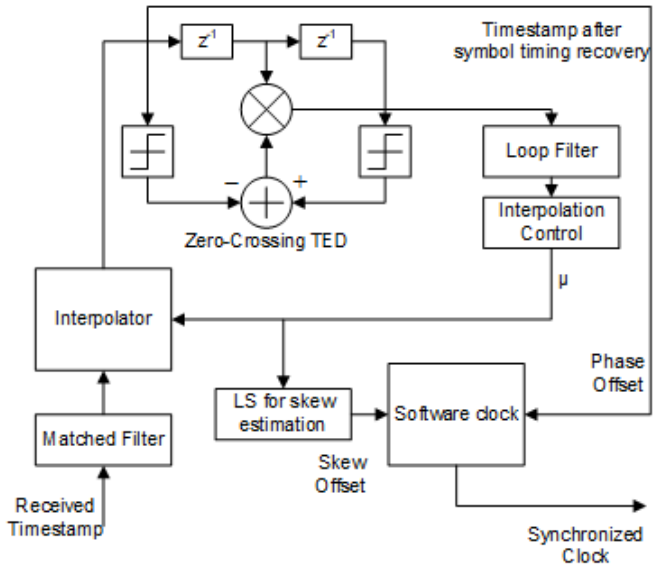

Figure 4. Single-timestamp skew-correction (STSC) model.

The output of the matched filter is shown in the eye diagram in Figure 5. The received symbols had a timing error. When that error was tracked using the symbol timing recovery method, the output of the interpolator was synchronized in time and the corresponding eye diagram is shown in Figure 6. This eye diagram shows that the PAM symbols were simply recovered. The scatter plot before (Figure 7) and after the timing recovery (Figure 8) also shows that without time synchronization, no pair of wireless nodes could communicate with each other.

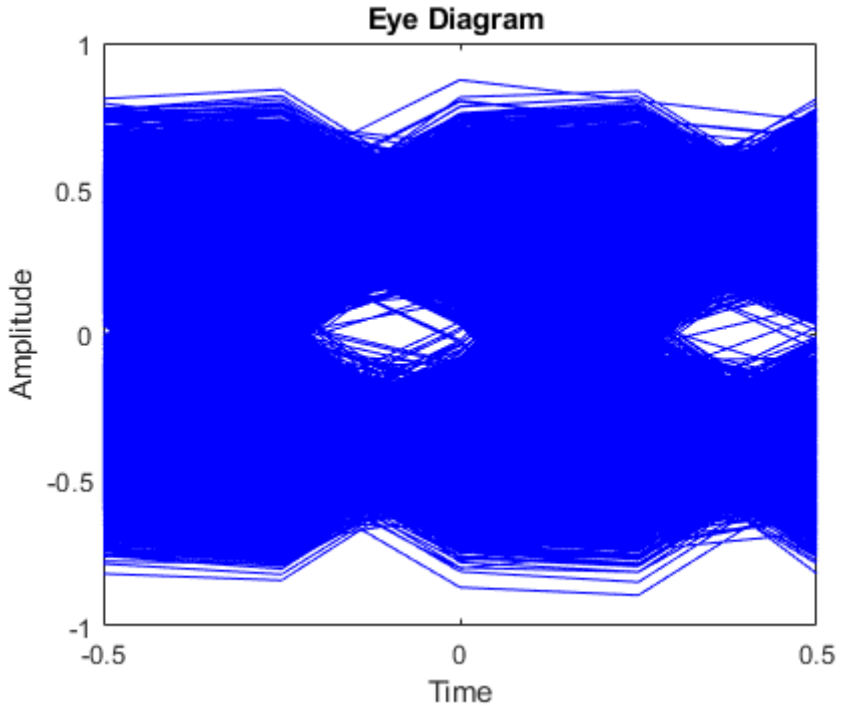

Figure 5. Eye diagram of received symbols without symbol timing recovery.

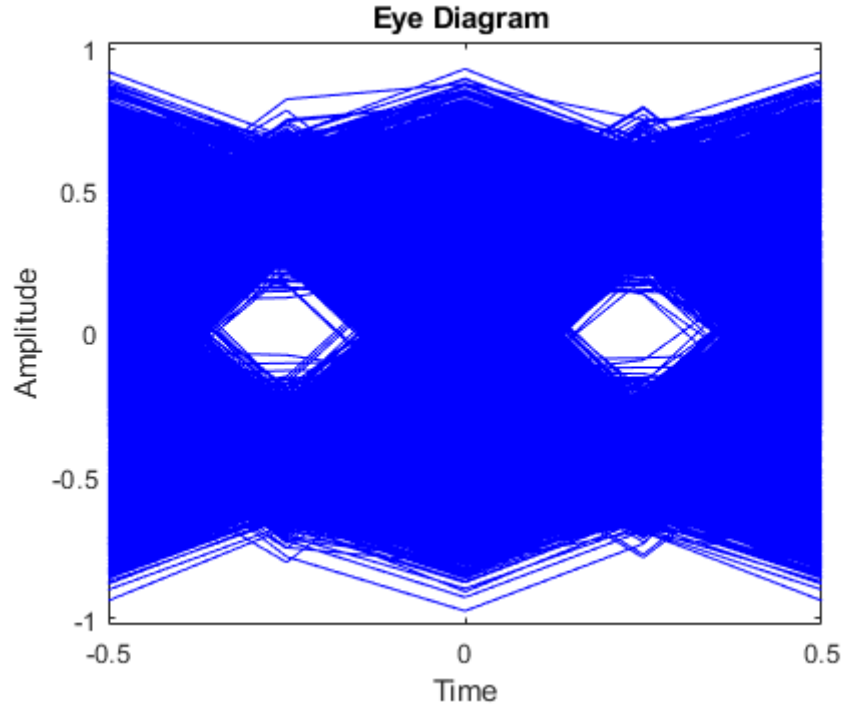

Figure 6. Eye diagram of received symbols after symbol timing recovery.

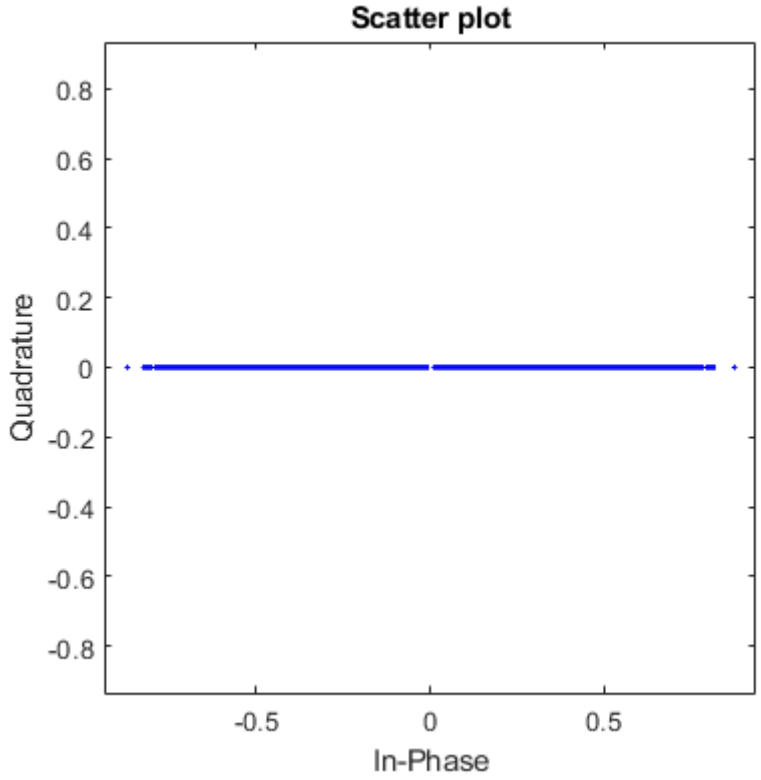

Figure 7. Scatter plot of received symbols without symbol timing recovery.

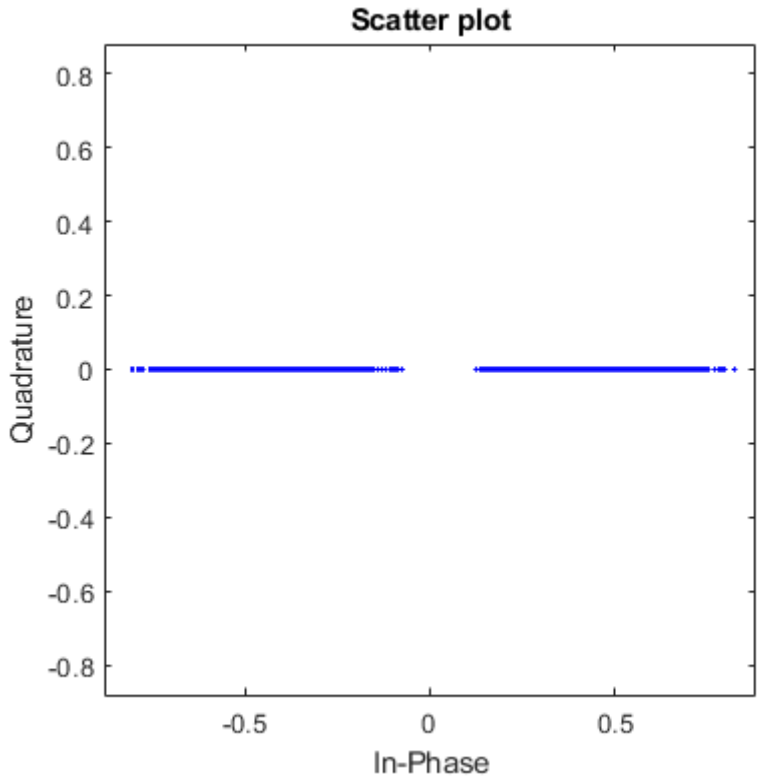

Figure 8. Scatter plot of received symbols after symbol timing recovery.

The plot of the fractional offset μ is shown Figure 9. This offset converged to one point and the least squares (LS) method was used to obtain the time skew. Applying the LS method to the fractional offset produced the graph shown in Figure 10. Hence, the computed output shows that the hardware clock skew offset was −3.1581 µs. Keeping this in mind, we then considered the software clocks running on node A and node B. Some of the values of both clocks are shown in Table 1. These values were 1000 ticks apart. For the first value, we computed the phase offset which was 19,000 and could be corrected by the timestamp packet. Considering one tick of the software clock was 1000 ticks of the hardware clock, the skew of the software clock was computed and yielded $1 - \frac{278{,}015 - 273{,}000}{259{,}000 - 254{,}000} = 0.003$ = 3 ms. The skew of the hardware clock computed by the STSC approach was 3.1581 µs and after translation to the software clock's skew, it was 3.1581μ × 1000 ticks = 3.1581 ms, which was approximately the same as that computed above. Hence, the software clock derived from this hardware clock had the same offset. The skew of the software clock could be adjusted using that offset and once done, the clocks were synchronized in frequency and the contents of the timestamp packet was used to synchronize in phase. The software and hardware clocks of the sender and receiver were then in synchronization using a single timestamp, as described for the STSC method.

Table 1. Values of software clocks after 1000 ticks.

Node A Clock	273,000	274,003	275,006	276,009	277,012	278,015
Node B Clock	254,000	25,500	256,000	257,000	258,000	259,000

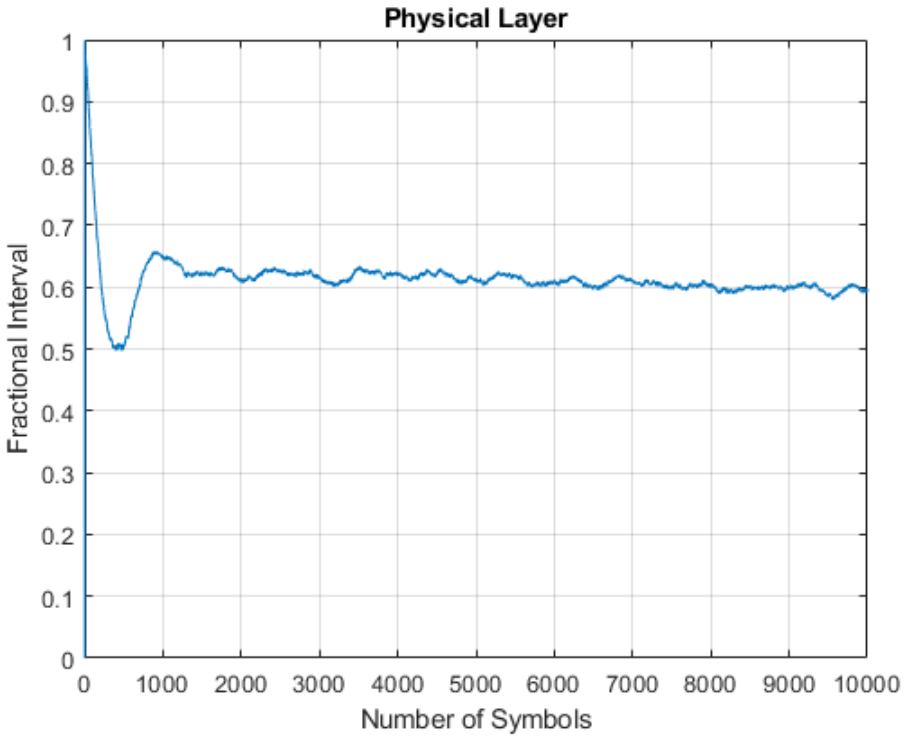

Figure 9. Fractional offset μ.

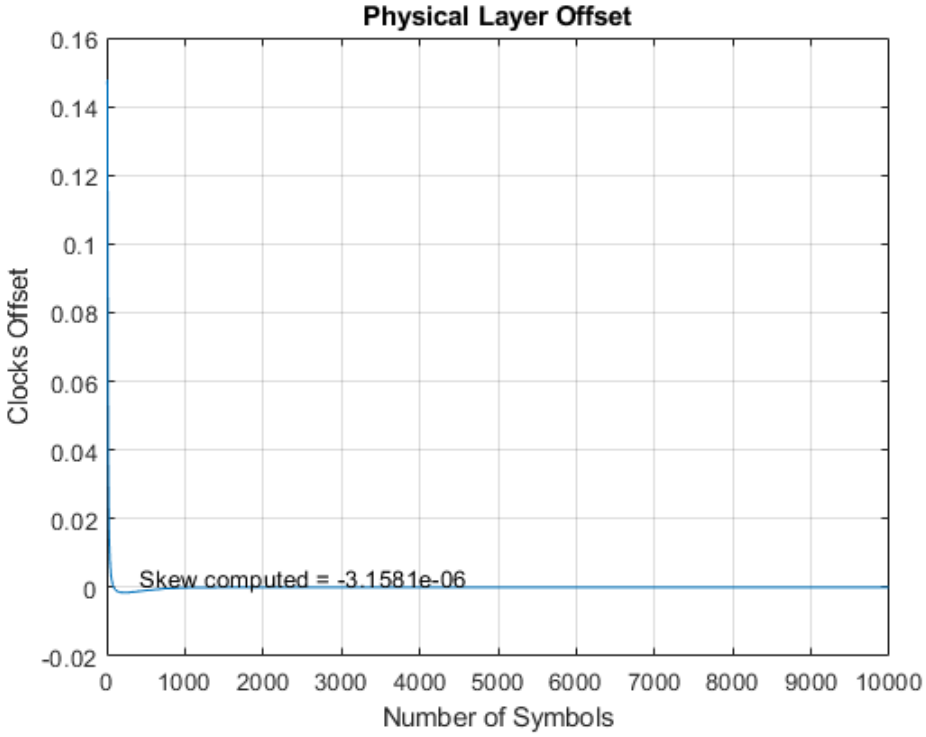

Figure 10. Computation of skew offset using the LS method.

The experiment was conducted again using two different DSKs with a known offset of around 3 µs. The STSC method computed the skew estimate for the new nodes, which was 3.2183 µs, as shown in Figures 11 and 12.

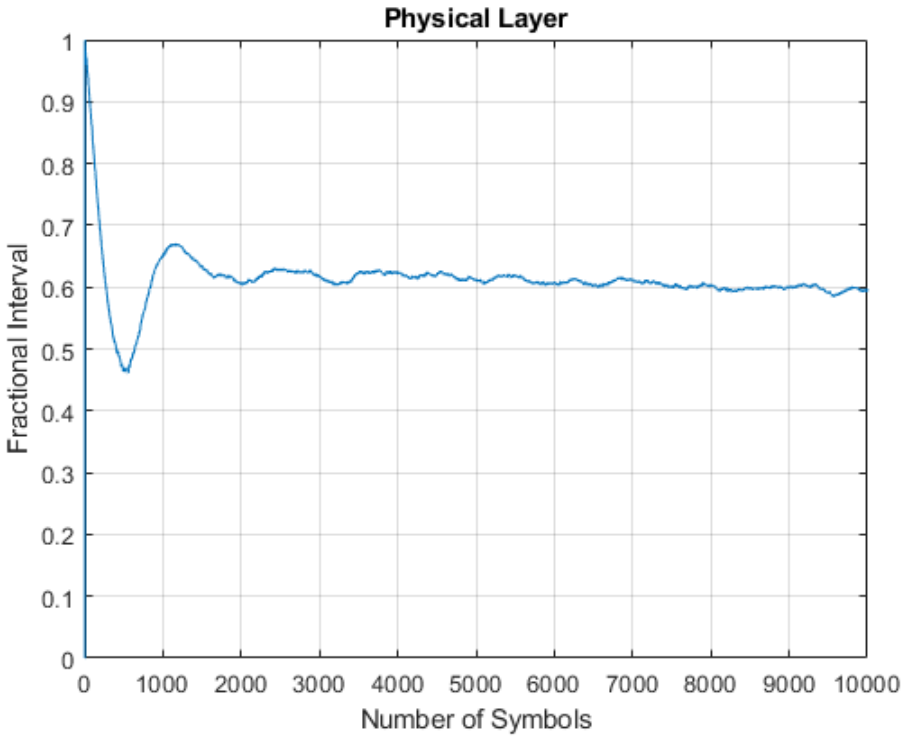

Figure 11. Fractional offset μ.

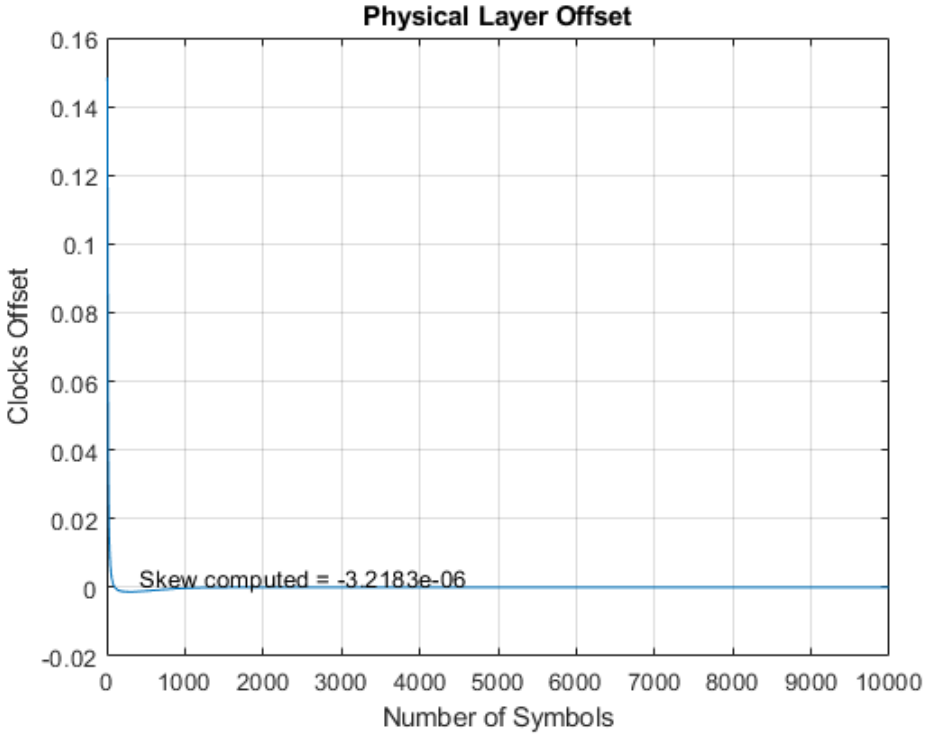

Figure 12. Computation of skew offset using the LS method.

To further strengthen the experimental results, the experiment was conducted a third time using two different DSKs with a known offset of around 20 µs. The STSC method computed the skew estimate for the new nodes as 20.907 µs, as shown in Figures 13 and 14.

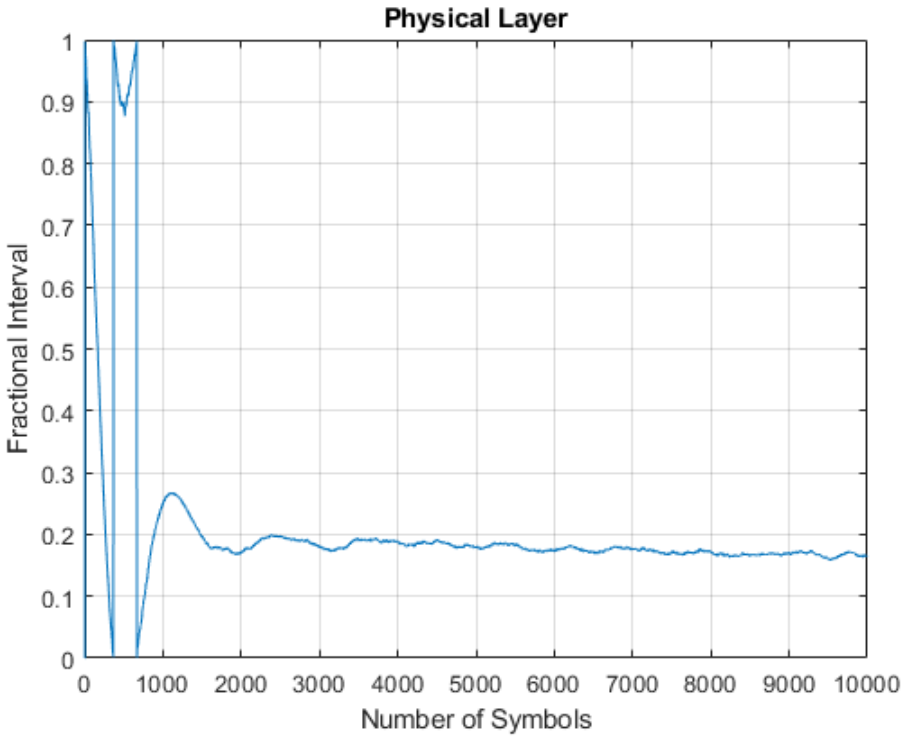

Figure 13. Fractional offset μ.

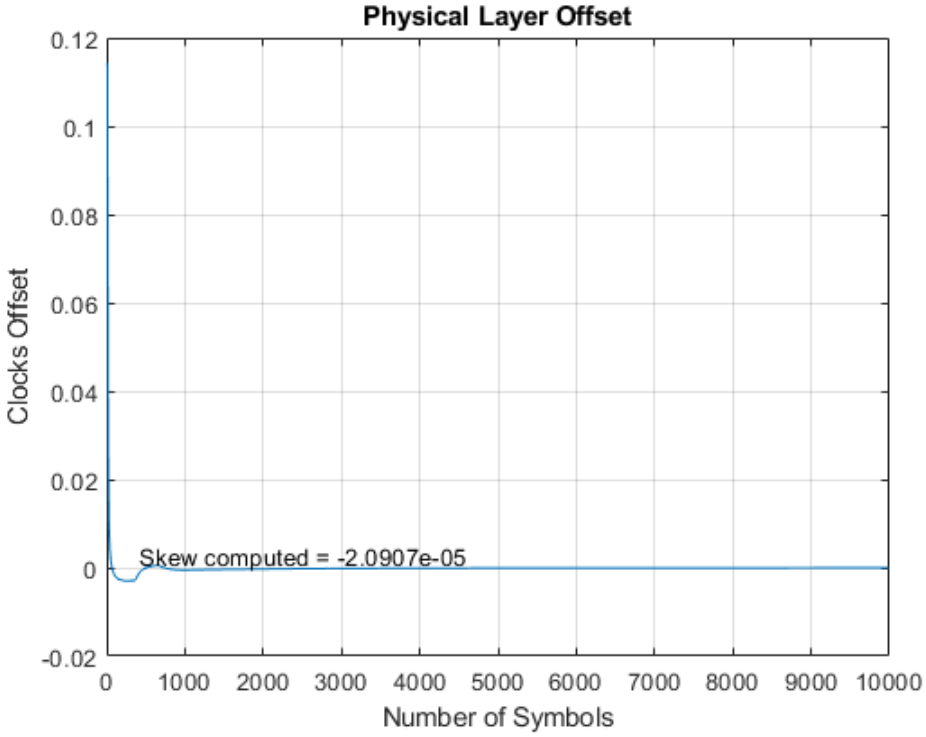

Figure 14. Computation of skew offset using the LS method.

In the literature, many different state-of-the-art time synchronization protocols, such as, flooding time synchronization protocol (FTSP), timing-sync protocol for sensor networks (FTSP), timestamp synchronization (TSS), and lightweight time synchronization (LTS) at the application layer were discussed in the related work section. All these protocols used a multiple-timestamp exchange in order to calculate the skew of the clock. They tended to

decrease the message exchanges by either incorporating timestamps in the data package (e.g., TTS), creating a spanning tree of the network nodes following a centralized algorithm (e.g., LTS), or using a level hierarchy (TPSN). Still, the timestamp exchanges required were more than one in the mentioned and other available protocols. In the STSC, only one timestamp was enough for the synchronization at the physical and application layers and for estimating the clock skew. For example, the FTSP synchronized two nodes with two timestamps, compared to the STSC method, which could synchronize with one timestamp, doubling the efficiency theoretically. In contrast to existing works, the proposed STSC approach was applied between any two nodes of the V2X system, and the time synchronization could be achieved with one timestamp packet, hence saving the transmission and computational energy required to send multiple timestamps.

6. Conclusions

This paper presented a skew-based approach, namely, a single-timestamp skew correction (STSC), for the time synchronization in V2X networks. The proposed method only required a single timestamp to estimate the time skew at the hardware level with the help of the physical-layer time synchronization using symbol timing recovery. The result showed that the skew computed at the hardware clock gave a sufficient estimate of the software clock's skew. In the STSC approach, one timestamp packet was used to correct the phase offset as well as the skew offset. The skew computed at the hardware clock was approximately the same as that of the software clock. We showed that only one packet proved to be sufficient to compensate for both phase and frequency offset. Saving multiple timestamp exchanges led to a greater energy conservation. However, we note that more STSC packets could be used to further enhance the accuracy if required for highly sensitive V2X systems.

Author Contributions: M.U.H. and M.H. conceived and planned the conceptualization, methodology, writing of the original draft preparation, and performed the experiments. M.B. and B.Q. contributed to the analysis and interpretation of the results, reviewing, editing, and funding acquisition. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Acknowledgments: The authors would like to acknowledge the support of Prince Sultan University for paying the Article Processing Charges (APC) of this publication.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

STSC	Single Timestamp Skew Correction
DSP	Digital signal processor
V2X	Vehicle-to-everything
V2P	Vehicle-to-pedestrian
V2I	Vehicles-to-infrastructure
V2C	Vehicle-to-cloud
WSN	Wireless sensor network
LTS	Lightweight time synchronization
TPSN	Timing-sync protocol for sensor networks
FTSP	Flooding time synchronization protocol
PLL	Phase Lock Loop
LS	Least Square
DSP	Digital signal processing
ZC-TED	Zero-Crossing Timing Error Detector (ZC-TED)

References

1. Abboud, K.; Omar, H.A.; Zhuang, W. Interworking of DSRC and cellular network technologies for V2X communications: A survey. *IEEE Trans. Veh. Technol.* **2016**, *65*, 9457–9470. [CrossRef]
2. Bayu, T.I.; Huang, Y.F.; Chen, J.K. Performance of Fuzzy Inference System for Adaptive Resource Allocation in C-V2X Networks. *Electronics* **2022**, *11*, 4063. [CrossRef]
3. Do, D.T.; Van Nguyen, M.S.; Voznak, M.; Kwasinski, A.; de Souza, J.N. Performance analysis of clustering car-following V2X system with wireless power transfer and massive connections. *IEEE Internet Things J.* **2021**, *9*, 14610–14628. [CrossRef]
4. Hasan, M.; Mohan, S.; Shimizu, T.; Lu, H. Securing vehicle-to-everything (V2X) communication platforms. *IEEE Trans. Intell. Veh.* **2020**, *5*, 693–713. [CrossRef]
5. Abbas, F.; Liu, G.; Fan, P.; Khan, Z. An efficient cluster based resource management scheme and its performance analysis for V2X networks. *IEEE Access* **2020**, *8*, 87071–87082. [CrossRef]
6. Naidu, P.V.; Dhanekula, M.B.; Almustafa, K.M.; Kumar, A.; Meerja, K.A.; Akkapanthula, S.H. Design and performance analysis of MAZE shaped quad port ACS fed tri band MIMO antenna for V2V and multi band applications. *AEU-Int. J. Electron. Commun.* **2021**, *134*, 153676. [CrossRef]
7. Almasoud, A.S.; Eisa, T.A.E.; Obayya, M.; Abdelmaboud, A.; Al Duhayyim, M.; Yaseen, I.; Hamza, M.A.; Motwakel, A. Coyote Optimization Using Fuzzy System for Energy Efficiency in WSN. *Comput. Mater. Contin.* **2022**, *72*, 3269–3281. [CrossRef]
8. MacHardy, Z.; Khan, A.; Obana, K.; Iwashina, S. V2X access technologies: Regulation, research, and remaining challenges. *IEEE Commun. Surv. Tutor.* **2018**, *20*, 1858–1877. [CrossRef]
9. Wang, J.; Shao, Y.; Ge, Y.; Yu, R. A survey of vehicle to everything (V2X) testing. *Sensors* **2019**, *19*, 334. [CrossRef]
10. Gyawali, S.; Xu, S.; Qian, Y.; Hu, R.Q. Challenges and solutions for cellular based V2X communications. *IEEE Commun. Surv. Tutor.* **2020**, *23*, 222–255. [CrossRef]
11. Hasan, K.F.; Wang, C.; Feng, Y.; Tian, Y.C. Time synchronization in vehicular ad-hoc networks: A survey on theory and practice. *Veh. Commun.* **2018**, *14*, 39–51. [CrossRef]
12. Wu, Y.C.; Chaudhari, Q.; Serpedin, E. Clock synchronization of wireless sensor networks. *IEEE Signal Process. Mag.* **2010**, *28*, 124–138. [CrossRef]
13. Kopetz, H.; Ochsenreiter, W. Clock synchronization in distributed real-time systems. *IEEE Trans. Comput.* **1987**, *100*, 933–940. [CrossRef]
14. Hasan, K.F.; Feng, Y.; Tian, Y.C. GNSS time synchronization in vehicular ad-hoc networks: Benefits and feasibility. *IEEE Trans. Intell. Transp. Syst.* **2018**, *19*, 3915–3924. [CrossRef]
15. Abbasi, M.; Shahraki, A.; Barzegar, H.R.; Pahl, C. Synchronization techniques in "device to device-and vehicle to vehicle-enabled" cellular networks: A survey. *Comput. Electr. Eng.* **2021**, *90*, 106955. [CrossRef]
16. Khan, U.A.; Lee, S.S. Distance-based resource allocation for vehicle-to-Pedestrian safety communication. *Electronics* **2020**, *9*, 1640. [CrossRef]
17. Bregni, S. Characterization and modelling of clocks. *Synchronization Digit. Telecommun. Netw.* **2002**, *1*, 203–281.
18. Römer, K. Time synchronization in ad hoc networks. In Proceedings of the 2nd ACM International Symposium on Mobile Ad Hoc Networking & Computing, Long Beach, CA, USA, 4–5 October 2001; pp. 173–182.
19. Elson, J.; Girod, L.; Estrin, D. Fine-grained network time synchronization using reference broadcasts. *ACM SIGOPS Oper. Syst. Rev.* **2002**, *36*, 147–163. [CrossRef]
20. Hashmi, S.U.; Hussain, M.; Muslim, F.B.; Inayat, K.; Hwang, S.O. Implementation of symbol timing recovery for estimation of clock skew. *Int. J. Internet Technol. Secur. Trans.* **2021**, *11*, 241–268. [CrossRef]
21. Hashmi, S.U.; Hussain, M.; Arshad, S.N.; Inayat, K.; Hwang, S.O. Energy efficient cross layer time synchronisation in cognitive radio networks. *Int. J. Internet Technol. Secur. Trans.* **2021**, *11*, 329–340. [CrossRef]
22. Huang, D.J.; Teng, W.C.; Yang, K.T. Secured flooding time synchronization protocol with moderator. *Int. J. Commun. Syst.* **2013**, *26*, 1092–1115. [CrossRef]
23. Sattar, D.; Sheltami, T.R.; Mahmoud, A.S.; Shakshuki, E.M. A comparative analysis of flooding time synchronization protocol and recursive time synchronization protocol. In Proceedings of the International Conference on Advances in Mobile Computing & Multimedia, Vienna, Austria, 2–4 December 2013; pp. 151–155.
24. Panayirci, E.; Bar-Ness, E.K. A new approach for evaluating the performance of a symbol timing recovery system employing a general type of nonlinearity. *IEEE Trans. Commun.* **1996**, *44*, 29–33. [CrossRef]
25. Kim, D.K.; Do, S.H.; Cho, H.B.; Chol, H.J.; Kim, K.B. A new joint algorithm of symbol timing recovery and sampling clock adjustment for OFDM systems. *IEEE Trans. Consum. Electron.* **1998**, *44*, 1142–1149.
26. Lee, D.; Cheun, K. A new symbol timing recovery algorithm for OFDM systems. *IEEE Trans. Consum. Electron.* **1997**, *43*, 767–775.
27. Wang, J.; Yang, Z.X.; Pan, C.Y.; Han, M.; Yang, L. A combined code acquisition and symbol timing recovery method for TDS-OFDM. *IEEE Trans. Broadcast.* **2003**, *49*, 304–308. [CrossRef]
28. Belega, D.; Fontanelli, D.; Petri, D. Dynamic phasor and frequency measurements by an improved Taylor weighted least squares algorithm. *IEEE Trans. Instrum. Meas.* **2015**, *64*, 2165–2178. [CrossRef]
29. Simeone, O.; Spagnolini, U.; Bar-Ness, Y.; Strogatz, S.H. Distributed synchronization in wireless networks. *IEEE Signal Process. Mag.* **2008**, *25*, 81–97. [CrossRef]

30. Leng, M.; Wu, Y.C. Distributed clock synchronization for wireless sensor networks using belief propagation. *IEEE Trans. Signal Process.* **2011**, *59*, 5404–5414. [CrossRef]
31. Zhou, H.; Li, M.; Wang, N.; Min, G.; Wu, J. Accelerating Deep Learning Inference via Model Parallelism and Partial Computation Offloading. *IEEE Trans. Parallel Distrib. Syst.* **2022**, *34*, 475–488. [CrossRef]
32. Zhou, H.; Wu, T.; Chen, X.; He, S.; Guo, D.; Wu, J. Reverse auction-based computation offloading and resource allocation in mobile cloud-edge computing. *IEEE Trans. Mob. Comput.* **2022**. [CrossRef]
33. Rice, M. *Digital Communications: A Discrete-Time Approach*; Pearson Education India: Chennai, India, 2009.
34. Yang, H.K.; Snelgrove, M. Symbol timing recovery using oversampling techniques. In Proceedings of the ICC/SUPERCOMM'96-International Conference on Communications, Dallas, TX, USA, 23–27 June 1996; Volume 3, pp. 1296–1300.
35. Bertolucci, M.; Cassettari, R.; Fanucci, L. On the frequency carrier offset and symbol timing estimation for CCSDS 131.2-B-1 high data-rate telemetry receivers. *Sensors* **2021**, *21*, 2915. [CrossRef]