An Approach for Precise, Scalable, and Platform Independent Clock Synchronization

Henning Puttnies, Dirk Timmermann
University of Rostock
Institute of Applied Microelectronics and Computer Engineering
18051 Rostock, Germany
Tel./Fax: +49 (381) 498-7277 / -1187251
Email: henning.puttnies@uni-rostock.de

Peter Danielis
ACCESS Linnaeus Center, School of Electrical Engineering
KTH Royal Institute of Technology, Stockholm, Sweden
Email: pdanieli@kth.se

***Abstract*—Clock synchronization is an important issue in wired and wireless networks as a common time basis is essential for coordinated activities of nodes in distributed systems. Typical applications demanding precise synchronization are Industrial Internet scenarios having real-time constraints, Wireless Sensor Networks (WSNs) where the nodes communicate only for a short period and can sleep the remaining time to save energy, approaches based on Time Division Multiple Access (TDMA), and distributed measurements. The basic idea of our approach is to estimate all delays in a network. As a result, we can estimate the one-way delay between a reference node and all other nodes in the network. Consequently, we can use broadcast messages to synchronize the entire network. Utilizing a novel measurement method and a Java prototype implementation, we show that our approach achieves a high precision ($\approx 123\ \mu s$). Furthermore, it is highly scalable and platform independent. As our synchronization approach operates at the application layer it is suitable for both wired and wireless networks.**

***Index Terms*—Synchronization, Internet of Things, Industrial Internet, Quality of Service, Network Latency**

I. Introduction

Clock synchronization, which is the provision of a common time basis, is crucial for coordinated activity in distributed systems: in Industrial Internet scenarios (Smart Factory, Smart Power Plant), where hard or soft real-time constraints must be met, a precise common time base is essential for an accurate action of distributed systems composed of sensors and actors. Furthermore, in Wireless Sensor Networks (WSNs), if the communication follows a fixed schedule all sensors that do not communicate at a specific point in time can fall into sleep mode and save valuable energy [1]. Additionally, a precise synchronization is important for all communication approaches based on Time Division Multiple Access (TDMA) as well as distributed measurements, where every sample that is measured by a specific sensor needs an exact timestamp to be comparable to samples from other sensors.

The Network Time Protocol (NTP) and the Precision Time Protocol (PTP) are the established standards for clock synchronization. NTP is implemented in Software, but is not very precise (approx. $1\ ms$) [1]. PTP can reach a higher precision ($<\ 1\ \mu s$), needing specialized HW. As based on hardware timestamps is it highly platform depended to reach this precision. Software implementations of PTP have much lower precisions [2].

Our approach bases on two considerations: the first consideration is that the estimation of the delay between two nodes N_0 and N_1 in a network and the synchronization of these nodes are equivalent problems. If we know the delay, we can take a $timestamp_0$ at node N_0, transmit it to node N_1 and adjust the local clock of node N_1 by a value that equals $timestamp_0 - delay$. On the other hand, if two nodes are perfectly synchronized we can send a $timestamp_0$ from node N_0 to node N_1 and take a second $timestamp_1$. The delay between N_0 and N_1 equals $timestamp_1 - timestamp_0$. The second consideration is that various unknown parameters compose the delay between two nodes. We can distinguish between software-introduced delays (e.g., packet processing on nodes) and hardware-introduced delays (e.g., packet processing in switches, propagation delays on wires). Whereas [3] describes that the software delays are variant and hard to estimate, we can assume that the hardware delays are less variant and much smaller (e.g., the delay in a switch is lower than 50 μs according to [4]).

Based on this considerations, we developed an algorithmic approach that reaches high precision, even though it is implemented in software, because it utilizes an estimation of the software delay. Moreover, our approach reaches a very good scalability, because the delays in the network need to be estimated only once and the synchronization as well as every resynchronisation can use one-way delays. This is a very strong point, as every network needs to be resynchronized after a specific period of time that depends on the precision of the clocks of the nodes [5]. The procedure of our approach is as follows:

1) Determination of the hardware delays in the network. These delays can be determined by measurements or from the specification of the switches as well as from the network topology using the Simple Network Management Protocol (SNMP).
2) We estimate the software delay of every node in the network by measuring several round-trip times (RTTs) to get a sufficient system of equations. Hereafter, we can estimate the software delay as a probability distribution.

3) We use the estimated delay for a synchronization based on one-way messages using the estimated one-way delay.

Compared to the state-of-the-art standards and research approaches, our method has the following advantages:

Scalability: if several nodes in the network have the same software delay (e.g., many identical sensors in an industrial network), we can reduce the number of packets to a theoretical minimum. There are no further delay estimations needed for the synchronization or the resynchronization. Instead, the reference node can use broadcast messages to synchronize all other nodes. Furthermore, every node has to save only one delay locally: the one-way delay between this node and the reference node.

High Precision: compared to other software-based approaches we reach a very high precision of the synchronization. In contrast to the state of the art, we can determine the software delay of a single node. In addition, we present a novel and very precise method to state the precision of our synchronization approach.

Plattform Independence: the proposed approach is independent of the physical layer (e.g., Ethernet, WIFI) as we estimate the introduced delays at the application layer. Moreover, we consider the software delay of the higher layers, which is important for latency-sensitive applications (e.g., Telemedicine). And finally, there is no need for expensive additional hardware.

The remainder of this paper is organized as follows: Section II introduces technical basics and established standards in the field of clock synchronization. Section III describes related research works. Section IV presents the proposed approach followed by Section V elaborating experimental results and their evaluation. Finally, we conclude the paper in Section VI, and present directions for future work.

II. Technological Basis

First, we will examine established standard protocols as well as technological basis as clock synchronization is a vital filed of research.

To allow for a precise synchronization among devices, the IEEE 1588 Precision Time Protocol (PTP) has been developed, which exists as both a hardware and a software version [2]. The PTP synchronization starts with determining, which of the devices has the most stable and accurate clock by means of the Best Master Clock algorithm (BMCA) and selects this device as master. Then, reference times are sent to the slaves, which compare them with their own time. On uniformly distributed intervals, slaves respond with their own time references to the master. From the differences of the answers, the master-to-slave delay and the slave-to-master delay can be determined to be able to synchronize devices. The hardware variant achieves an accuracy of nanoseconds in the best case. However, the main disadvantages of the hardware variant are that dedicated hardware has to be available and time stamps are taken in lower layers thereby ignoring the delay introduced by higher layers. The existing software version takes time stamps in the application layer and achieves an accuracy on the order of microseconds with a hardware-supported master clock. As opposed to our approach, this version is subject to several hardware and software restrictions and hence cannot be considered as platform independent. Furthermore, solely the sum of the software delays of two devices can be determined and it is impossible to distinguish between the particular delays contributing to the RTT.

The latter disadvantages applies to the Network Time Protocol (NTP) as well [6]. NTP represents a standard that enables the synchronization of clocks on devices in a distributed system. Again, the principle of reference clocks is used. Clients send a packet to the server at certain points in time and wait for the answer. From the time of arrival of the reply, the packet RTT can be calculated. From it, the latency can be estimated assuming symmetric propagation delays. In the public Internet, an accuracy of better than 10 ms can be achieved with this protocol. At network level, there exists a hierarchy of participating servers leading to a hierarchical structure of time references. Thus, the accuracy can be decreased due to possible error propagation along this hierarchy.

Wu et al. survey developments for the clock synchronization of WSNs by exchanging timestamps between sensor nodes [7]. More specifically, the authors introduce the fact that due to the imperfections of the clock oscillator, a clock will drift away from the ideal time even if it is initially perfectly tuned. Thus, the relative clock offset keeps changing with time and therefore the network has to perform periodic clock resynchronization to adjust the clock parameters. Moreover, the work discusses network-wide synchronization. As opposed to the synchronization between a pair of neighboring nodes, network-wide synchronization makes use of a hierarchical structure such as a tree and the synchronization is performed between adjacent levels of this hierarchy.

In [8], the authors investigate to which extent clock synchronization is even feasible. Thereby, they consider clocks at a constant but not necessarily identical speed. Each clock is an affine clock characterized by its skew and an offset with respect to a reference clock. To establish fundamental impossibility results, the unknown parameters skew and offset are considered to be constant time-invariant parameters. The main finding is that the skews can be determined correctly but the determination of all clock offsets and link delays is impossible without further simplification.

III. Related Work

In [9], the authors present an overview of IEEE 802.1AS, which is part of a set of standards originally developed by the Audio/Video Bridging (AVB) Task Group. In 2012, this task group has been renamed as Time-Sensitive Networking (TSN) Task Group. The focus of 8021.1AS is on exact timing and synchronization, reservation of resources and traffic shaping, and queuing as well as data forwarding. Basically, IEEE 802.1AS bases its synchronization on IEEE 1588 PTP, uses a modified version of PTP's BMCA, and achieves an accuracy of ± 500 ns relative to the grandmaster clock. Each port of a

so-called time-aware system measures the propagation delay to its neighboring time-aware systems in a way, which is mathematically equivalent to the manner the IEEE 1588 peer-to-peer transparent clock transports synchronization. A time-aware system is referred to as a bridge or end-station that meets the requirements of IEEE 802.1AS whereby all bridges and end-stations in the 802.1AS network are required to be such time-aware-systems.

As the accuracy of skew and offset estimations, the stability of the slave clocks as well as the exchange intervals of timing information affect the synchronization performance, Giorgi et. al analyze the effects and interplay of these factors in [10]. The objective is to show how these may impact the design of an IEEE 1588 synchronization scheme. The analysis is based on a state-variable simulation model, which has been shown to be able to model specific aspects of clock behavior. Further, in these models Kalman filters are useful for the implementation of the clock servo. Although a Kalman filter may not always be realizable due to possible limitations of PTP devices like the availability of floating-point processing, such a solution improves synchronization capabilities since it provides optimal estimates in the presence of measurement noise.

In [11], various measures are analyzed to mitigate asymmetric delays with IEEE 1588 PTP without protocol changes and any additional messages. The author proposes to correct timestamps to the reference node at each egress and ingress port by the device driver. Their software approach cannot reach the precision of a hardware timestamping approach but anyway can almost eliminate the clock offset in a single-hop WLAN synchronization systems.

Nilsson et al. present a statistically robust method for passive clock synchronization in sensor networks [12]. To measure delays, messages are time stamped at the measuring node, transmitted to a central node, and time stamped there again with respect to the central clock. The timing is then estimated by means of exploiting the heavy tailed likelihood function to neglect outliers. Contrary to a Kalman filter, the synchronization becomes more precise this way as outliers basically falsify the Kalman filter estimates. The approach works fine for a sequence of measurements by excluding large outliers but does not investigate measures to avoid a central node as a single point of failure.

In [13], a protocol called PulseSync is presented, which aims at synchronizing large-scale networks. This protocol floods the network with rapid, short pulses to be able to have a short initialization phase and to quickly adapt to changes regarding topology or drift. Thereby, nodes minimize their offset and skew towards a root node. It is shown that PulseSync outperforms the de facto standard protocol Flooding Time Synchronization Protocol (FTSP) for WSNs. However, the authors mention that they have to improve the robustness of their protocol as the root node constitutes a single point of failure. Also, as timestamps are taken at the MAC layer, no consideration of the delay introduced by higher layers is possible.

Mallada et al. propose a network clock synchronization protocol without the need for skew estimation in [14] and show the protocol's supremacy over some existing solutions like NTP. Timestamps are based on the Time Stamp Counter (TSC) that counts the number of CPU cycles since the last restart and can provide the timestamp at the application layer. The time measurements are then carried out using an improved ping pong mechanism [15]. The algorithm uses the obtained current offset information and an exponential average of past offsets to avoid keeping track of long offset history and expensive computations on them. However, the authors propose an algorithm comprising estimations of the clock offset between every node and the neighbors of this node. For a network of n nodes this introduces at least n estimations (assuming every node has only one neighbor) for every resynchonization. In contrast, the time for resynchronizations based on our approach is independent of the number of nodes in the network.

A major enhancement compared to the state of the art is the scalability of our approach: in the existing protocols, for the execution time T the following equation holds: $T \sim n$ (n = number of nodes in the network), if all nodes are synchronized with one reference node. Alternatively, $T \sim log(n)$, if we have a tree structure, with many reference nodes. Nevertheless, in a tree structure the error is also proportional to $log(n)$ due to error propagation. In contrast, our approach scales with m (m = number of unique platforms, $m \leq n$). Thus, we need the time $T \sim m$ to estimate the software delays of all unique platforms in the network only once. Hereafter, we can use broadcast messages for synchronization and resynchronization. Thus, we need a Time $T \sim 1$ that is independent of the number of nodes for every (re-)synchronization.

IV. Approach

The proposed synchronization algorithm bases on the estimation of all delays in the network. Afterwards, we can estimate the one-way delay between the reference node and all other nodes in the network. Considering the delays, especially the software delay is variant and hard to estimate [3]. Therefore, we will focus on the estimation of the software delay in the following.

A. Generating a Solvable System of Equations

As shown in Figure 1, if we estimate the RTTs separately, we cannot distinguish between the delay d_0 of the node N_0 and the delay d_1 of the node N_1 both contributing to the RTT. In contrast, if we employ a system of equations, it is possible to estimate every delay separately. This leads to a more precise estimation as the software delay is variant. Therefore, we estimate one probability distribution for the software delay of a particular node as it is less complex to estimate one probability distribution compared to the estimation of the sum of two probability distributions.

Every node in the network has a particular transmission delay d_t as well as a particular receipt delay d_r. To generate a solvable system of equations, we assume that the transmission delay d_t of a node is equal to the receipt d_r of this node. This simplification is essential to get a solvable system of equations.

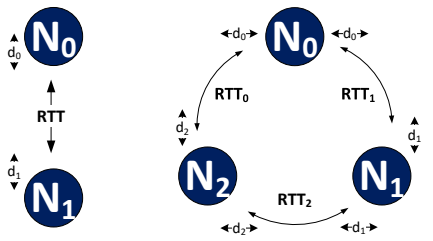

Fig. 1. Estimating the delay of network nodes using one RTT (left) or a system of equations (right)

Otherwise, it is impossible to determine all unknown parameters as shown in [8]. Furthermore, we can precisely state the error that this simplification introduces. If we have n nodes in the network, we need n independent equations for a system of equations of $rank\ n$. Hence, we first measure the $n-1$ RTTs between the reference node and the $n-1$ other nodes in the network. Thereafter, we measure one additional RTT between two nodes where none of these nodes is the reference node. Since the delay of every node contributes twice to every RTT, we get the following system of equations for three nodes (corresponding to Figure 1):

$$\begin{bmatrix} RTT_0 \\ RTT_1 \\ RTT_2 \end{bmatrix} = \begin{bmatrix} 2 & 2 & 0 \\ 2 & 0 & 2 \\ 0 & 2 & 2 \end{bmatrix} \cdot \begin{bmatrix} d_0 \\ d_1 \\ d_2 \end{bmatrix} + \begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix}. \quad (1)$$

RTT_x is the x-th RTT that is measured, d_x is the delay of the node N_x and c_x is the approximately constant hardware delay. For a network of n nodes, the system of equations is as follows:

$$\begin{bmatrix} RTT_0 \\ RTT_1 \\ \vdots \\ RTT_n \end{bmatrix} = \begin{bmatrix} 2 & 2 & 0 & \dots & 0 \\ 2 & 0 & 2 & & \vdots \\ \vdots & \vdots & & \ddots & 0 \\ 2 & 0 & \dots & 0 & 2 \\ 0 & 2 & 2 & 0 & \dots \end{bmatrix} \cdot \begin{bmatrix} d_0 \\ d_1 \\ \vdots \\ d_n \end{bmatrix} + \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{bmatrix}. \quad (2)$$

When we measure the constant hardware delay once, we can precisely estimate the software delay of a particular device by solving this system of equations.

B. Synchronization Procedure

We conduct the synchronization of all nodes in a network by executing several steps:

1) We select a reference node N_0. The easiest way to select this node is to introduce a random initial sleep time and select the node that awakes first as reference node. An alternative way to select the reference node is using a BMCA similar to PTP, which determines the node having the most precise clock.
2) The reference node N_0 estimates the RTTs to all other nodes (N_1,N_2). As we assume that the software delay is variant, the RTT is measured and estimated afterwards (e.g., by calculating the mean RTT).
3) The reference node N_0 sends a token to one of the other $n-1$ nodes (e.g., N_1).
4) The receiver of the token (e.g., N_1) estimates one further RTT to an arbitrary node (e.g., N_2) and returns the result back to the reference node (N_0).
5) The reference node solves the introduced system of equations utilizing the hardware delay as well as the estimated RTTs.
6) The reference node sends the hardware delays and the software delays of all nodes to the other $n-1$ nodes. Consequently, every node can calculate the one-way delay between itself and the reference node.
7) The reference node synchronizes the entire network by sending packets to all other nodes. These packets contain a $timestamp_0$ taken at the reference node and every node N_x takes a $timestamp_x$ after the receipt of the packet. The clock offset between the reference node N_0 and node N_x amounts to $timestamp_0 - timestamp_x - delay$ where $delay$ is the calculated one-way delay. Again, as we assume the software delay to be variant the clock offset is estimated by calculating the mean of multiple measurements.
8) For every resynchronisation, there is no need to estimate the delay again (assuming a time-invariant system). Thus, all nodes restart with step 6.

C. Conceptual Consideration of Error, Resiliency, and Scalability

Error: we can state the maximal error that arises from the assumed equality of the receipt and transmission delay: $Error_{max} = d_x$. Where d_x is the estimated software delay of a node x. Although, we cannot state the exact values of the transmission and receipt delay, we can precisely state the sum of both that equals $2 \cdot d_x$. Furthermore, we can formulate the upper bound for the worst case transmission and the worst case receipt delay of this node: $Delay_{worst} \leq 2 \cdot d_x$.

Resiliency: there is no single point of failure in the algorithm as a dynamic selection of the reference node is possible. Hence, the synchronization is still possible if we have at least three nodes in the network.

Scalability is an important aspect in the IoT scenarios of the future as there will be thousands of devices, which have to be connected. The proposed method exhibits good scalability: for the time T required to estimate the software delays of n nodes in a network, the following formula applies:

$$T \sim n. \quad (3)$$

We always need n measurements to get a system of equations of $rank\ n$ composed by n independent equations. However, the concept allows to reduce the number of unknowns. The

number of unknowns does not necessarily equal the number of nodes in the network. Instead, different nodes having identical platforms introduce identical delay distributions. Therefore, the number of unknowns equals the number of unique platforms in the network. We define a platform P_x as a unique pair of a hardware H_y and a software S_z:

$$P_x = (H_y, S_z). \tag{4}$$

Furthermore, we define the hardware as the quantity of all hardware components (processor, memory, network interface controller, ...) of a node and the software as the quantity of all software components (operation system, drivers, applications, network stack, ...) of this node.

$$H_y = \{Processor, NIC, Memory, ...\} \tag{5}$$

$$S_z = \{OS, Application, Drivers, ...\} \tag{6}$$

Consistently if two nodes have the identical software and hardware setup, they must show the same delay distribution. Even for thousands of nodes in an IoT scenario, we can assume that the quantity of unique platforms is much smaller than the number of nodes (e.g., if we have many sensors of the same vendor in a smart factory application). If we can assume time-invariance of the delay, we need only one estimation for every platform. Consequently, every resynchronization can use broadcast messages and thus the execution time for the resynchronization is independent of the number of nodes. We cannot assume the clock offset and skew of a node to be time-invariant, which leads to the need of resynchronizations. Nevertheless, we can assume the distribution of the software delay to be time-invariant utilizing a statistical sufficient estimation. Moreover, if new platforms join the network we only need to estimate the software delay of these.

V. Measurements and Evaluation

For the evaluation of our approach, we developed a Java prototype implementation. A simulation would not be reasonable as many parameters contributing to the software delay are unknown and thus have to be determined in a real test-bed first. We implemented the approach in Java to achieve high platform independence. In [16] the authors show that Java can even meet hard real-time requirements.

As experimental setup, we used three Intel Galileo boards [17] and a 1 GBit/s switch to connect these devices. To measure the precision of our synchronization, we developed a novel method to determine the offset between two clocks. Figure 2 exhibits the results of the FPGA-based reference measurement. We use an FPGA to generate a square-wave signal with a period of approx. $0.67\ s$ ($2^{26} \cdot 10\ ns$). The rising edges of this signal are time stamped by two nodes (the Galileo boards) and the timestamps are compared with each other. Although the time stamping of a GPIO pin introduces a software delay, we mitigate this influence by averaging over multiple timestamps as we assume that the jitter of the time stamps is introduces by the software processing of the GPIO signal and not by the FPGA. The clock offset calculated using these time stamps is exhibited in blue in Figure 2. Furthermore, we subtract out the clock skew that was approximately $21\ \mu s \mp 1\ \mu s$ per period ($0.67\ s$). Additionally, we average over the skew-compensated clock offset and thus calculate a precise reference value. As the last step, we compare the calculated reference value for the clock offset of the two nodes with the estimation done by our software implementation. We applied the measurement twice: we achieve a difference of $386\ \mu s$ between the FPGA-based reference value and the software estimation if we have an open SSH connection to all devices that affects the synchronization. Moreover, we achieved a difference of only $123\ \mu s$ using an isolated network without any interference. In both measurements the software uses only 10 RTTs to estimate the software delay and the clock offset.

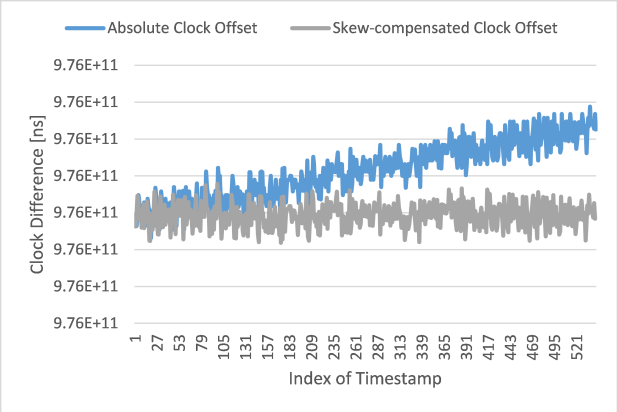

Fig. 2. FPGA-based reference measurement of the clock offset between two nodes

To evaluate the scalability of our approach, we emulate 100 nodes on one physical device utilizing the loopback IP Address. We generate a single thread for every node where all nodes use the same IP Address (loopback) but a different UDP port number. Figure 3 shows that the execution time is proportional to the number of unique platforms in the network. To estimate all n unknown parameters (the software delay of every unique platform), we need a system of equations of $rank\ n$. Therefore we have to estimate n RTTs and assume that the execution time is proportional to the number of unique platforms, which is lower or equal to the number of nodes. In contrast to the state-of-the-art, our approach needs to estimate the delay only once for every unique platform. Every resynchronization can use broadcast messages and is therefore independent of the number of nodes or unique platforms. Moreover, we examine the synchronization time in dependence of the number of RTTs used to estimate every unknown parameter. Figure 3 shows that the execution time is proportional to the number of RTTs used for every estimation. If we use more RTTs this results into a more precise estimation of the software delays and thus, a more precise synchronization. The number of packets needed for a statistical sufficient estimation of the software delay depends on the delay distribution of this platform. Hence, our approach supports the adaption of precision and execution time to the

requirements of the application.

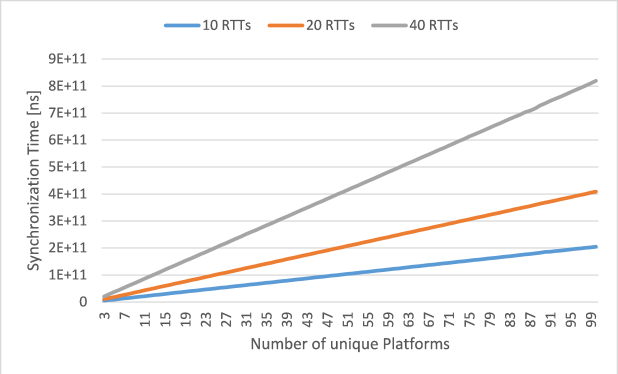

Fig. 3. Emulation-based investigation of scalability: synchronization time as a function of the number of unique platform in the network and the number of RTTs used for every estimation

VI. Conclusion and Future Work

In this paper, we present a highly precise, highly scalable, and platform independent approach for clock synchronization in networks, based on the estimation of all unknown delays in the network. We exhibit precision measurements where our approach outerperforms all existing software-based synchronization approaches. Furthermore, we expose that our approach outerperforms all exsiting methods in terms of scalability since we need to estimate the software delay only once and all further synchronizations and resynchronizations can use broadcast messages being independent of the number of nodes in the network. In our future work, we will consider the compensation of the clock skew as well as a precise estimation of the hardware delay. Furthermore, we will investigate several methods (e.g., Kalman filters) to get even more precise delay estimations.

Acknowledgment

The authors would like to thank the German Research Foundation (DFG) (research fellowship, GZ: DA 1687/2-1) for their financial support.

References

[1] M. Akhlaq and T. R. Sheltami, "Rtsp: An accurate and energy-efficient protocol for clock synchronization in wsns," *IEEE Transactions on Instrumentation and Measurement*, vol. 62, no. 3, pp. 578–589, 2013.

[2] K. Correll and N. Barendt, "Design considerations for software only implementations of the ieee 1588 precision time protocol," in *In Conference on IEEE 1588 Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*, 2006.

[3] A. Beifus, D. Raumer, P. Emmerich, T. M. Runge, F. Wohlfart, B. E. Wolfinger, and G. Carle, "A study of networking software induced latency," in *2015 International Conference and Workshops on Networked Systems (NetSys)*, pp. 1–8.

[4] E. Schweissguth, P. Danielis, C. Niemann, and D. Timmermann, "Application-aware industrial ethernet based on an sdn-supported tdma approach," in *2016 IEEE World Conference on Factory Communication Systems (WFCS)*. IEEE, 2016, pp. 1–8.

[5] I. Bojic and K. Nymoen, "Survey on synchronization mechanisms in machine-to-machine systems," *Engineering Applications of Artificial Intelligence*, vol. 45, pp. 361–375, 2015.

[6] D. Mills, J. Martin, J. Burbank, and W. Kasch, "Network time protocol version 4: Protocol and algorithms specification," Tech. Rep., 2010.

[7] Y.-C. Wu, Q. Chaudhari, and E. Serpedin, "Clock synchronization of wireless sensor networks," *IEEE Signal Processing Magazine*, vol. 28, no. 1, pp. 124–138, 2011.

[8] N. M. Freris, S. R. Graham, and P. Kumar, "Fundamental limits on synchronizing clocks over networks," *IEEE Transactions on Automatic Control*, vol. 56, no. 6, pp. 1352–1364, 2011.

[9] M. D. J. Teener and G. M. Garner, "Overview and timing performance of ieee 802.1 as," in *2008 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*. IEEE, 2008, pp. 49–53.

[10] G. Giorgi and C. Narduzzi, "Performance analysis of kalman-filter-based clock synchronization in ieee 1588 networks," *IEEE Transactions on Instrumentation and Measurement*, vol. 60, no. 8, pp. 2902–2909, 2011.

[11] R. Exel, "Mitigation of asymmetric link delays in ieee 1588 clock synchronization systems," *IEEE Communications Letters*, vol. 18, no. 3, pp. 507–510, 2014.

[12] J.-O. Nilsson and P. Händel, "Robust recursive network clock synchronization," in *Electronics, Computing and Communication Technologies (IEEE CONECCT), 2014 IEEE International Conference on*. IEEE, 2014, pp. 1–5.

[13] C. Lenzen, P. Sommer, and R. Wattenhofer, "Pulsesync: An efficient and scalable clock synchronization protocol," *IEEE/ACM Transactions on Networking (TON)*, vol. 23, no. 3, pp. 717–727, 2015.

[14] E. Mallada, X. Meng, M. Hack, L. Zhang, and A. Tang, "Skewless network clock synchronization without discontinuity: Convergence and performance," *IEEE/ACM Transactions on Networking*, vol. 23, no. 5, pp. 1619–1633, 2015.

[15] S. Froehlich, M. Hack, X. Meng, and L. Zhang, "Achieving precise coordinated cluster time in a cluster environment," in *2008 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*. IEEE, 2008, pp. 54–58.

[16] B. Konieczek, M. Rethfeldt, F. Golatowski, and D. Timmermann, "Real-time communication for the internet of things using jcoap," in *2015 IEEE 18th International Symposium on Real-Time Distributed Computing (ISORC)*, pp. 134–141.

[17] "Intel galileo board." [Online]. Available: https://www.arduino.cc/en/ArduinoCertified/IntelGalileo