Settling Issues in IEEE 802.1AS Networks in PI Based Clock Servos

Attila Frankó[‡], Gergely Hollósi[‡]
[‡]*Dept. of Telecommunications and Media Informatics*, *Faculty of Electrical Engineering and Informatics*
Budapest University of Technology and Economics, Műegyetem rkp. 3., H-1111 Budapest, Hungary
{attila.franko, hollosi.gergely}@vik.bme.hu

***Abstract*—Time-sensitive networking (TSN) is a set of standards for establishing real-time, deterministic communication in industrial applications. One of the core tasks of TSN is providing accurate and precise time synchronization between nodes by using a profile for the Precision Time Protocol (PTP). This profile defined by IEEE 802.1AS standard – often called gPTP – is restrictive regarding the synchronization method, only allowing peer-to-peer (P2P) synchronization protocol between the grand-master (GM) and slaves. Most use cases apply PI controller as a servo, which affects the overall accuracy and precision. Here we show that the increasing number of intermediate hops between the slave and the GM significantly deteriorates the synchronization accuracy. Based on a closed-loop model, we simulated the behavior of the slave clock during multi-hop P2P synchronization by investigating the step response of the system. These results were validated by measurements in real network setups using embedded devices as endpoints. Our results demonstrate that such errors cannot be completely eliminated using PI controller even if optimal tuning is used; therefore, researching and implementing other methods are recommended.**

***Index Terms*—tsn, time synchronization, IEEE 802.1AS, gptp, linuxptp, pi controller**

I. Introduction

Communication plays a key role in industrial automation, especially nowadays when numerous applications in digital production systems require real-time communication capabilities [19]. There are plentiful areas in such digital industrial ecosystems, e.g. Industrial Internet of Things (IIoT), Cyber-physical Systems (CPS), Manufacturing Execution Systems (MES) which demand hard real-time data exchange with additional timing and reliability requirements between nodes [18], [9]. These application fields are not entirely new, however most of them have been widely adopted only in the last few years, therefore, addressing the communication requirements still is a hot and intensively researched topic.

In the last two decades different classes of Real-Time Ethernet (RTE) networks became the most prevalent ways to achieve industrial communication with the hard-real time requirements between devices. Recently, the biggest driver in this field is shifted from streaming data (audio-video) to control data thus motivating the creation of Time-sensitive Networking (TSN) [20]. TSN is a set of various standards to extend Ethernet in order to achieve real-time synchronization and deterministic, low-latency communication [4], [17].

One of the pillars of TSN is standard IEEE 802.1AS [6], Timing and Synchronization for Time-Sensitive Applications. It utilizes Precision Time Protocol (PTP, IEEE 1588 [5]) as the mechanism for synchronizing devices, but the standard itself is more restrictive regarding several criteria, than PTP itself thus only enabling a specific profile for the protocol, often called general(ized) PTP (gPTP).

gPTP profile has numberous restrictions including the exclusion of end-to-end (E2E) measurements, allowing only peer-to-peer (P2P) measurements between nodes. This completely excludes the usage of transparent clocks, which means that slaves cannot obtain the grand-master's clock directly by adjusting the measured path delay between them – only if they share a direct link; otherwise, they have to synchronize to the next boundary clock [10].

In this paper, we model synchronization with a closed-loop control system to describe how the PI controllers affect the P2P path delay mechanism. The impact of the controllers will be demonstrated by simulations and validated by real-network measurement setups. The results show how the number of nodes hops between the GM and the slave and also the parameters of the controllers affect the overshoot and the settling time of the overall system.

The paper is structured as follows. Section III introduces the control system model of the synchronization method and the potential issues supported by simulations. Section IV details the measurement setups and the applied tools and equipment, moreover it presents the results of the measurements and compares them to the simulated ones. Section V concludes the paper and suggests further research directions within the topic.

II. Related work

There are different ways to implement a servo that is responsible for keep the local clock in sync with the master, however most applications including *linuxptp* [1], the user space PTP stack for Linux implements the servo as a PI (Proportional and Integral) controller.[1] In end-to-end measurements this choice moderately affect the overall performance of the control due to usage of transparent clocks - which enables delay request-response mechanism through the device. Therefore, in the

[1]Theoretically, the user can choose the servo type, but currently only PI and an adaptive controller using linear regression are implemented.

E2E case only the endpoint will run the servo in order to synchronize to the master clock, while in the P2P case each node runs the servo as all of them must keep the precise time synchronicity.

There are several articles that modelled PTP synchronization process differently as these models focus on different aspects of process. In terms of control theory it is usually described as a discrete control loop (PI controller, plant) with additional filters and estimators to minimize error [15], although not only PI controller can be used as a servo. Authors of [8] propose Kalman-filter while in [16] a linear programming approach is introduced, while in [7] open-loop algorithm is investigated and evaluated that comes from the classical closed-loop model. However, these solutions performed very well in simulated environments (or in test measurements as in [14], where authors proposed an adaptive fuzzy-PI servo), practical use-case almost exclusively utilize PI servo as it is a fairly simple servo and is available in all PTP applications (PC bridges, switches, embedded devices, etc.) – which is one of the most important factor and the reason we investigate a PI servo model in this paper.

In [13] authors proposed a discrete model with a PI servo to describe the synchronization protocol. They derived the stability criteria of the system from the state transition matrix and also showed the stability region on the $k_P - k_I$ plane. In [21] a Kalman-filter based algorithm is introduced but as a connector to a PI servo to reduce the growth rate of synchronization error due to the quantization error in timestamping, which is the result of casacded networks where the number of hops has a traceable impact on the performance of synchronization.

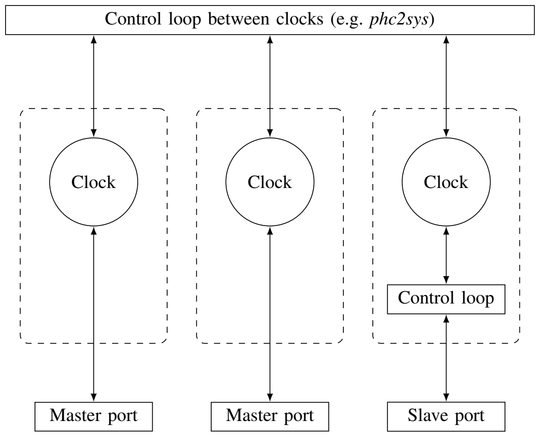
Since we investigate the P2P synchronization of any node in the network, we must distinguish between the P2P synchronization model and the E2E model in which there is no additional node between the grand-master and the slave. Usually, there is no differentiation in PTP related literature, as E2E measurements are allowed, but in this case additional nodes might change the characteristics of the communication significantly.

Because of the restriction of the 802.1AS standard, no transparent clocks are allowed, but only ordinary (OC) and boundary clocks (BC). An ordinary clock shall contain a single PTP port, while a boundary clock shall contain multiple ones. It also means that ordinary clocks have one servo while boundary clocks can have as many as PTP ports they have. Usually, switches have one local clock as boundary clock, while PC bridges have as many PTP port they have, since most Network Interface Cards have one hardware clock for all of their ports. While transparent clocks transit time information and corrects propagation delay, non-transparent boundary clocks can transmit only their own time information. It requires the corresponding PTP ports to be in sync which can be achieve by *phc2sys* software, but not exclusively. In a general sense, synchronization over a non-transparent boundary clock can be performed by after it is synchronized to the GM and also the two ports must be sychronized as well, which may imply the usage of two servos, especially in the case of PC bridges. Eventually, this will affect the proposed P2P control system model [10], [5].

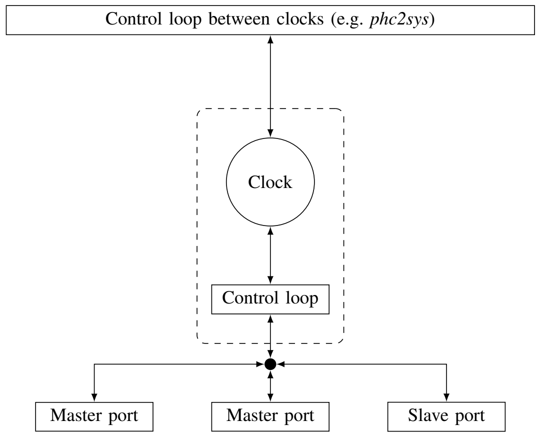
III. Methods

A. Model

In this paper, we model the synchronization process between a master and a slave – as it's seen in Figure 2 – with one PI controller and the plant to be controlled – which is the slave clock. While prior articles such as [12] also introduced a comparable model, it is important to note that their model is discrete and more general. In contrast, this paper employs a continuous model, aligning with the approach of *linuxptp* in considering controller parameters as continuous.



(a) Boundary clock with distinct physical clocks per port synchronized by an additional control loop



(b) Boundary clock with shared local clock

Fig. 1: Simplified models of an shared local clock and separated local clock OC or BC according to the IEEE 1588 standard [5]

The model is built on the commonly used closed-loop base model that consist of (i) one PI controller (ii) one plant with a transfer function of $W_p(s) = 1/s$, that represent the the clock to be synchronized and (iii) the negative feedback of the response to the output. In P2P based synchronization process,

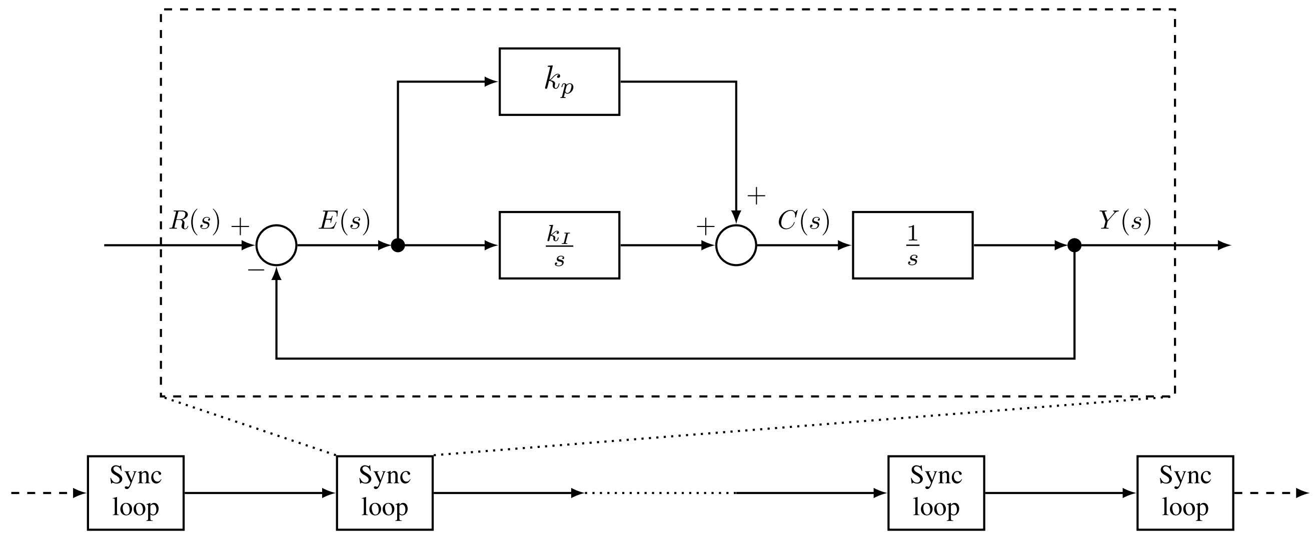


Fig. 2: Block diagram of the closed-loop E2E and P2P master-slave synchronization model, where a block is consisting of one PI controller and the slave clock

N control loops are chained, where the transfer function of the ith control loop is

$$W_c^{(i)}(s) = \frac{k_P^{(i)}s + k_I^{(i)}}{s^2 + k_P^{(i)}s + k_I^{(i)}} = \frac{2\zeta^{(i)}T_0^{(i)}s + 1}{s^2(T_0^{(i)})^2 + 2\zeta T_0^{(i)}s + 1} \tag{1}$$

with dumping factor $\zeta^{(i)}$ and time-constant $T_0^{(i)}$ as

$$\zeta^{(i)} = \frac{k_P^{(i)}}{2\sqrt{k_I^{(i)}}} \quad T_0^{(i)} = \frac{1}{\sqrt{k_I^{(i)}}} \tag{2}$$

The nature of the synchronization implies that the $r(t)$ reference signal is not static, in fact it is increasing linearly as it is the time of the grand-master clock. Nevertheless, the error caused by the dynamic reference is usually negligible compared to the one coming from the cascading second-order systems, which can be adjusted by choosing appropriate values of $k_P^{(i)}$ and $k_I^{(i)}$ as it is shown in Equation 2.

The P2P measurement indicates that the underlying P2P control system model is a serialization of multiple master-slave models with the transfer function

$$W(s) = \prod_{i=1}^{N} W_c^{(i)}(s) \tag{3}$$

However, we assume that each closed-loop is the same, the communication between each node has the same characteristics and also that each controller has the same set of parameters, i.e. $T_0^{(i)} = T_0$ and $\zeta^{(i)} = \zeta$. In this case, the overall transfer function is

$$W(s) = \frac{(2\zeta T_0 s + 1)^N}{(s^2T_0^2 + 2\zeta T_0 s + 1)^N} \tag{4}$$

resulting in the original poles of $W_c(s)$ with a multiplicity of N. This multiplicity ends in more and more amplification at the frequency specified by the time constant, effectively decreasing the efficiency of the synchronization.

In Figure 1 different boundary clocks are shown: one has distrinct physical clock for each port, while other one has only one shared clock that is used by all ports. Therefore, in the first case, if the number of the nodes between the slave and the grand-master is n, then the P2P model consists of $2n + 1$ cascaded E2E models since the bridge nodes between the GM and the end station (slave) are non-transparent boundary clocks with distinct physical clocks, therefore and additional internal synchronization between their PTP ports is required.

B. Simulations

In order to illustrate the effect of the parameter adjustments, we will use a system described by the E2E model with the default parameters of *linuxptp*: $k_p = 0.7$ and $k_I = 0.3$. According to the bode plot of the frequency response – shown in Figure 4 – there is significant amplification around the system's natural frequency that's given by $\omega_0 = 1/T_0 = 0.5477\,\mathrm{rad\,s^{-1}}$. The amplification will increase in the P2P model as it's composed of multiple cascaded E2E ones, thus having a great impact on the behavior of the system.

The simulated step responses of the cascaded model for certain values of n can be seen in Figure 3. According to these results, even several nodes between the grand-master and the slave can significantly decrease the accuracy of the synchronization due to the overshoot occurring in the responses.

By tuning the parameters of the controller, the amplification around the natural frequency can be damped, thus flatting the frequency response and decreasing the oscillations, however this will affect not only the overshoot but the settling time of the system. The T_s settling time is usually meant as the time when the response reaches and stays within a range of 2% of

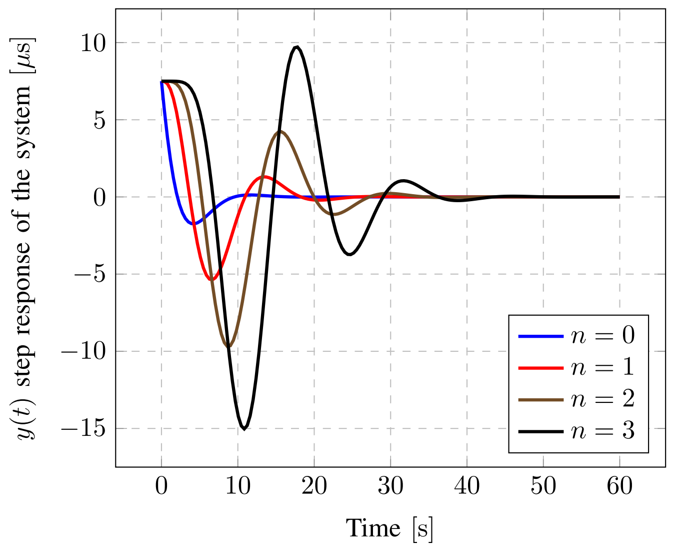


Fig. 3: Simulations of the model's step responses to P2P measurements, in the case of $n = \{0, 1, 2, 3\}$ intermediate nodes between the GM and the slave using the *linuxptp*'s default k_p and k_i values

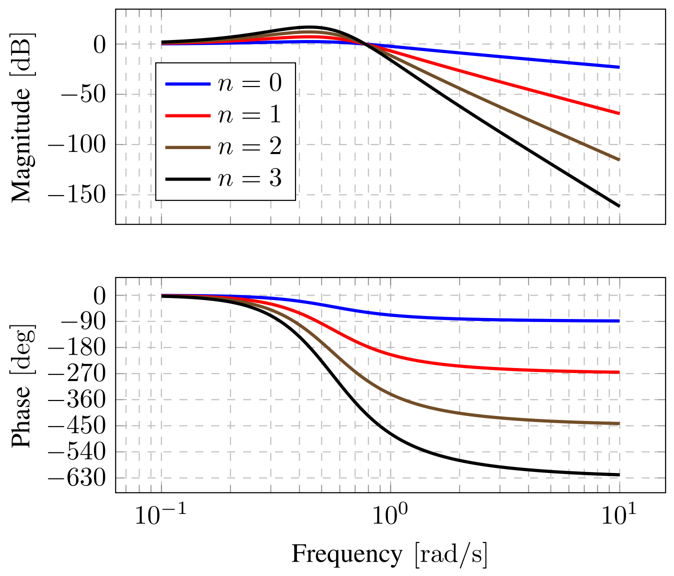


Fig. 4: Bode plot of the system's frequency response for $n = \{0, 1, 2, 3\}$

the steady state value and is often expressed by the ζ damping factor as shown in Equation 5.

$$T_s = \frac{-\ln\left(0.02 \times \sqrt{1-\zeta^2}\right)}{\zeta\omega_0} \tag{5}$$

As Equation 5 presents, the damping factor directly affects the settle time: the bigger the damping factor, the slower the settling, and vice versa. It is worth noting that changing the damping implies a change in the natural frequency, so the dependency is not linear.

Consequently, considering any clock that uses a PI controller as a servo, the dumping factor of the system can be used to adjust its response to meet the requirements regarding the overshoot or settle-time. However, it must be emphasized that limiting the overshoot will significantly increase the settle-time as it's stated before and shown in Figure 5 and Figure 6. Due to the nature of the system, an optimal tuning for the parameters can always be found, as in [13] was shown that a stable combination of k_P and k_I always exists for such systems as the proposed E2E model – nevertheless as the n increases the overall properties of the system deteriorate despite the optimal settings.

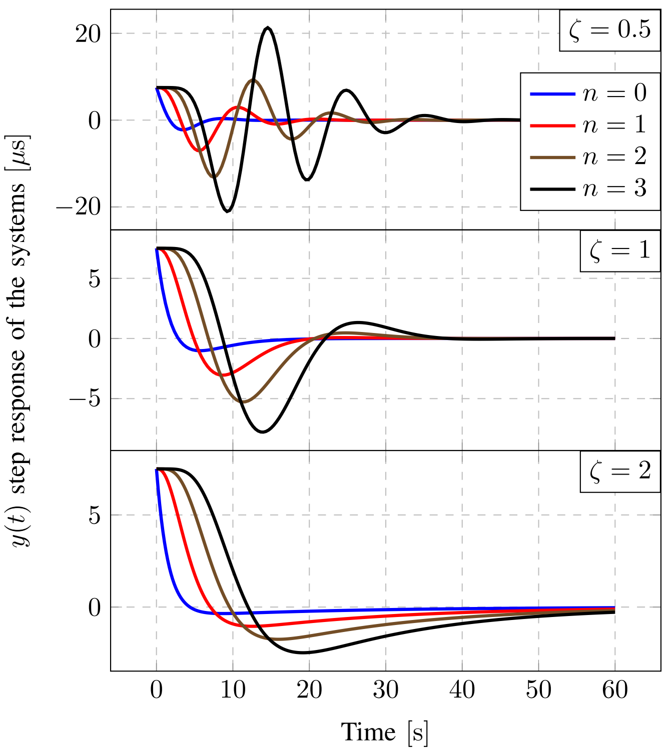


Fig. 5: Simulation results of the model's step responses to P2P measurements, in the case of $n = \{0, 1, 2, 3\}$ for different damping factor (ζ) values – the legend is shared between the plots

In the next sections these results will be verified and supported by real-network measurements in different setups.

IV. Evaluation

A. *Measurements*

To prove the correctness of the model proposed in Section III and to investigate the overall behavior of a multi-node network that gPTP compliant, multiple measurement environments have been set up. Two desktop PC-s (TSN, NST) were used as bridge machines i.e. intermediate nodes between the GM and the slave, both with different configurations:

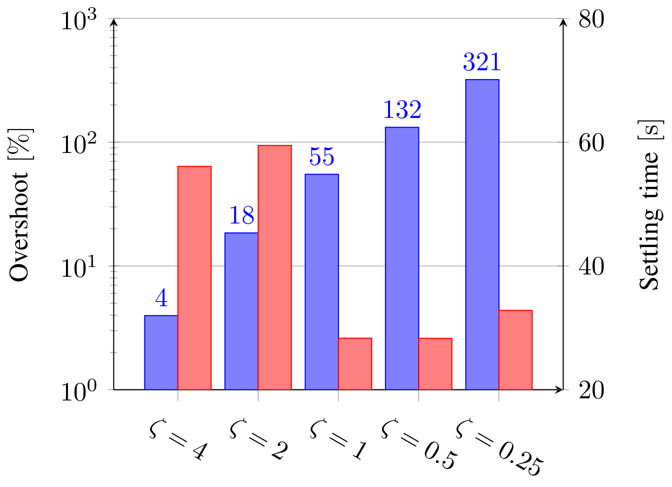


Fig. 6: Overshoot and settling time values of simulation results in the case for $n = 3$ for different damping factor (ζ) values

- TSN bridge:
  - ASUS PRIME Z270M-Plus Motherboard
  - Intel(R) Core(TM) i7-7700K CPU @ 4.20GHz
  - I350 Gigabit Network Connection and Ethernet Controller 10-Gigabit X540-AT2 2-port Network Interface Controller connected via PCIe 4x slots
  - 8 Gbytes of RAM
  - Ubuntu Linux with kernel 4.13.0-32-generic
- NST bridge:
  - ASUS Z9PE-D8 WS 1.0x Motherboard
  - Intel(R) Xeon(R) CPU E5-2620 v2 @ 2.10GHz
  - 2 pieces of 82574L Gigabit Network Connection 1-port Network Interface Controller connected via PCIe 4x slots
  - 16 Gbytes of RAM
  - Ubuntu Linux with kernel 5.4.0-126-generic

The measurements rely on the comparison of Pulse-per-second (PPS) signals of the GM and the slave devices, but none of the NIC-s have easily available PPS output – neither supported by the driver nor physically available on a separated pin. Therefore two embedded STM32F407 devices with an Ethernet extension shield were used as GM and slave, since the device enables precise time-stamping for PTP messages and the chip itself supports PPS signals.

The firmware running on these devices is our port to STM32F407 of an open source PTPd implementation for STM32F429 [3] boards that was already used and verified in scientific researches [11].

In each setup, the PPS signals from the GM and the slave were connected to STM32G431 based embedded device, which also calculated the difference of the signals. PPS signals serve as triggers on two individual GPIO pins that are mapped to the input capture function of a general-purpose timer, therefore, if the rising edge of a PPS signal has been captured, the corresponding channel identifier and timer value will be stored. For this purpose, a 32-bit resolution timer with a frequency of 195.5 MHz was used. This implies that the resolution of the measurement is approx. 5.9 ns, while 25.34 second difference (which is more than 25 PPS interval) can be held by the timer.

In the first (DI) setup, as shown in Figure 7, the GM and the slave devices were connected directly via Ethernet, thus the previously defined E2E model can be validated. For this setup, the default parameters of the embedded PTPd implementation were used, and no adjustments were applied.

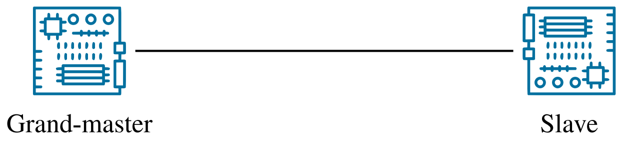


Fig. 7: Network setup for $n = 0$, direct measurements – the difference meter is not included in these diagrams as it's not connected to any device via Ethernet

In the following setups (M1S1, M1S2, M1S3) are peresented in one diagram in Figure 8. The GM and the slave were connected via a bridge node (TSN or NST) because of two reasons: (i) inspect the behaviour of different NIC-s that might affect the overall results and (ii) to validate the simulation for $n = 1$.

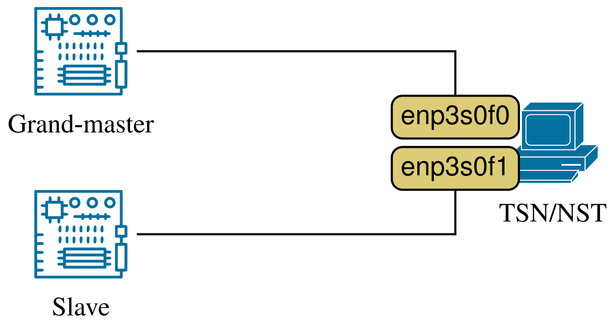


Fig. 8: Network setup for $n = 1$, P2P measurements – it covers all cases (M1S1, M1S2, M1S3) as only the NIC-s were changed

The following setups involves $n = 2$ and $n = 3$ intermediate nodes between the GM and the slave, respectively. Bridge devices use *linuxptp* (*ptp4l*) in each setup.

Due to the limitations of the embedded PTPd implementation and in order to avoid complicated network setups, all P2P measurements are achieved by direct and internal synchronizations: Two neighboring nodes perform a synchronization process over gPTP and thus synchronizing the clocks of the link, while hardware PTP clocks of the ports (that can be on the same NIC but also can be on different ones) are synchronized internally by using the *phc2sys* software.

In each case, the step response of the closed-loop should be determined, therefore step function has to be mixed with

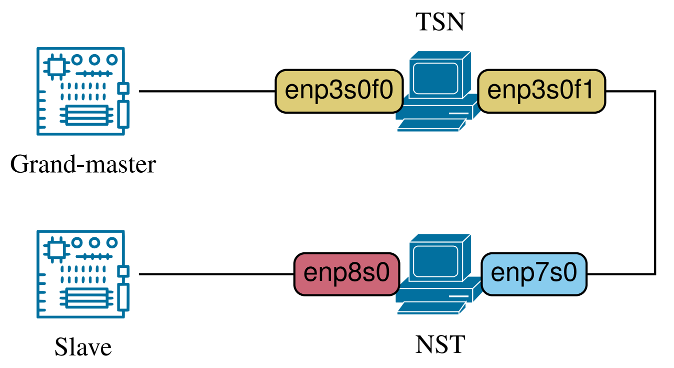


Fig. 9: Network setup for $n = 2$, P2P measurements – same color means the same, physical NIC, but different ports. As it's seen NST bridge has only 1-port NIC-s, but the two NIC-s are identical

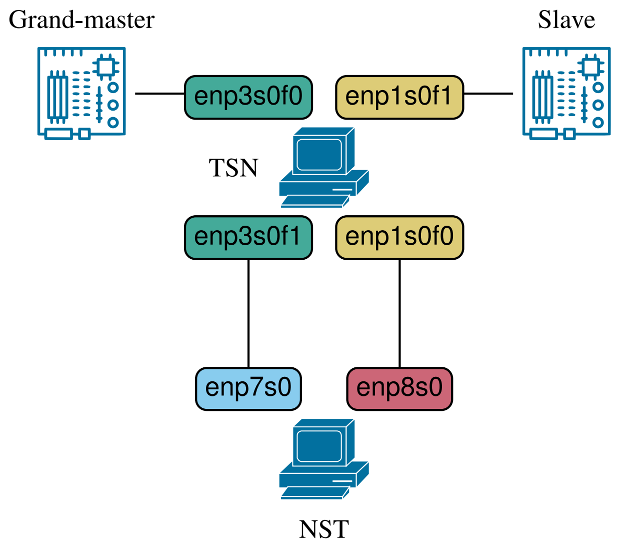


Fig. 10: Network setup for $n = 3$, P2P measurements

the clock signal as the $r(t)$ reference function. Since, the $H(t)$ unit step function cannot be used, as the PI controller cannot follow such magnitudes in offset difference – and resets the clock instead – the used step size is $\Delta = 7.5\,\mu s$. Regarding our measurement setup, we can not determine the $y(t)$ response signal since only the difference of PPS signals can be measured, i.e. the $e(t)$ error signal.

B. Results

As it's stated above, the measurements focus on the $e(t)$ error functions for the different setups. Nevertheless, since the $r(t)$ reference signal is the sum of a quasi-linear clock signal of the GM and a step function, we can approximate the response as follows: $y(t) = \Delta \cdot H(t) - e(t)$, where $H(t)$ is the step function, Δ is the step size. Therefore, the characteristics of the closed-loop – overshoot, settle time – can be determined based on only the $e(t)$ signal itself.

As it's shown in Figure 11, the response to the clock offset of the NIC-s used in the bridge PCs are very similar. Despite not having an optimized controller configuration, in the stationary state, the overall precision is around 10 ns interval and the settling time can be considered fast. This performance is achieved by using Precision Time Measurement (PTM) [2], which enables the coordination of timing between multiple devices on the same (PCIe) bus. Without PTM, several microseconds of delays would have a big impact on the accuracy of the overall system.

According to the measurement results, there is no significant difference between the NICs regarding the settling time and the overshoot, however in the case of M1S1, significant ringing occurred. This effect does not have a huge impact on the response of the system, but it makes difficult to determine the settling-time and the steady-state value. Nonetheless, regarding the most important properties of the step responses, the NIC can be considered almost identical.

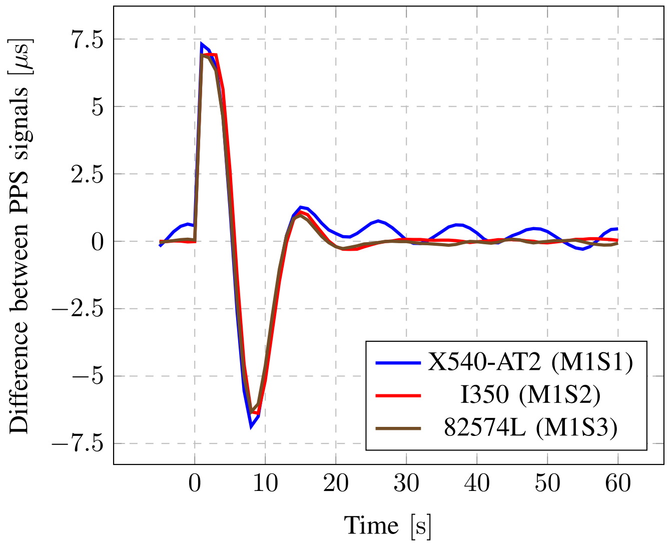


Fig. 11: Step responses of the system while using different Network Card Interfaces as the intermediate node – the time axis has been relabeled to make $t = 0$ denote the clock step

It can be seen by comparing Figure 3 and Figure 12 that the results of the multi-node setup measurements follow the simulations: the overshoot and the settle-time significantly increase with number of n. However, in this case, the k_I and k_p parameters of the servo are the default values provided by the *linuxptp*, thus they are not adjusted to the given plant. This fact does not affect the overall characteristics of the response since it is a second-order system, as it was stated before.

	$n = 0$	$n = 1$	$n = 2$	$n = 3$
Overshoot [%]	31	97	214	341
Settling time [s]	12	24	30+	30+

TABLE I: Overshoot and settling time values based on the measurements results in the case of $n = \{0, 1, 2, 3\}$

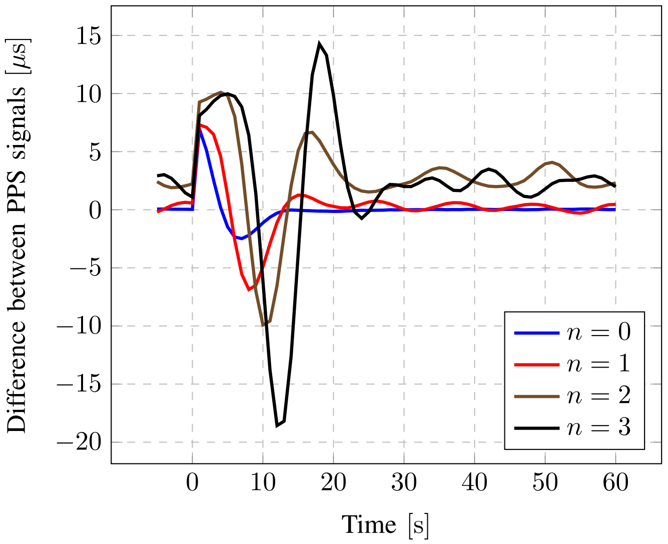


Fig. 12: Step response of the system during P2P measurements, in the case of $n = \{0, 1, 2, 3\}$ intermediate nodes between the GM and the slave – the time axis has been relabeled to make $t = 0$ denote the clock step

V. Conclusions

This paper analyses the P2P synchronization method used in the IEEE 802.1AS (gPTP) profile for PTP protocol regarding the servo mechanism. Since PI (Proportional and Integral) controller is the prevalent servo used in all PTP implementations, we used a classic closed-loop to model the synchronization process of two neighboring (directly connected) nodes. Since P2P synchronization prohibits propagating the grand-master (GM) clock as a boundary clock, slaves can synchronize only to the neighboring nodes. Therefore, the full sync process between a slave and the GM can be modeled by cascaded closed-loop (assuming the identical plant and controller settings).

In order to validate the model above, both simulations and measurements over a real network were carried out. Our measurement results follow the simulation as if the number of the intermediate nodes increases, then the overshoot and the settle-time of the overall system increase as well, degrading the synchronization performance significantly. Since the model is based on a second-order system, it is impossible to eliminate this performance degradation completely, however with proper parameter tuning, the response can be optimized to meet specific criteria regarding the overshoot or the settle-time of the system. Further research activities involve investigating other servo mechanisms (e.g. previously proposed only for PTP or not proposed at all) to achieve more effective synchronization control in gPTP-enabled TSN networks.

References

[1] The Linux PTP project. https://linuxptp.nwtime.org. [Online; accessed 12-aug-2023].

[2] Precision time measurement. https://www.intel.com/content/www/us/en/docs/programmable/683140/21-4-4-0-0/precision-time-measurement-ptm-58323.html. [Online; accessed 14-aug-2023].

[3] PTPd for STM32F4 boards. https://github.com/mpthompson/stm32_ptpd. [Online; accessed 6-aug-2023].

[4] https://www.ddc-web.com/en/support/technical-support/white-papers/a-deterministic-ethernet-solution-based-on-time-sensitive-networking-tsn, 2020. [Online; accessed 14-July-2023].

[5] Ieee standard for a precision clock synchronization protocol for networked measurement and control systems. *IEEE Std 1588-2019 (Revision ofIEEE Std 1588-2008)*, pages 1–499, 2020.

[6] Ieee standard for local and metropolitan area networks–timing and synchronization for time-sensitive applications. *IEEE Std 802.1AS-2020 (Revision of IEEE Std 802.1AS-2011)*, pages 1–421, 2020.

[7] Carsten Andrich, Julia Bauer, Peter Große, Alexander Ihlow, and Giovanni Del Galdo. A fast and stable time locked loop for network time synchronization with parallel fll and pll. In *2018 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)*, pages 1–4, 2018.

[8] Giada Giorgi and Claudio Narduzzi. Performance analysis of kalman-filter-based clock synchronization in ieee 1588 networks. *IEEE Transactions on Instrumentation and Measurement*, 60(8):2902–2909, 2011.

[9] Zeba Idrees, Jose Granados, Yang Sun, Shahid Latif, Li Gong, Zhuo Zou, and Lirong Zheng. Ieee 1588 for clock synchronization in industrial iot and related applications: A review on contributing technologies, protocols and enhancement methodologies. *IEEE Access*, 8:155660–155678, 2020.

[10] Michael D. Johas Teener and Geoffrey M. Garner. Overview and timing performance of ieee 802.1as. In *2008 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*, pages 49–53, 2008.

[11] Kaiyao Lao and Gangfeng Yan. Implementation and analysis of ieee 1588 ptp daemon based on embedded system. In *2020 39th Chinese Control Conference (CCC)*, pages 4377–4382, 2020.

[12] D. Macii, D. Fontanelli, and D. Petri. A master-slave synchronization model for enhanced servo clock design. In *2009 International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*, pages 1–6, 2009.

[13] Ryuichiro Maegawa, Daiki Matsui, Yasuhiro Yamasaki, and Hiroyuki Ohsaki. A discrete model of ieee 1588-2008 precision time protocol with clock servo using pi controller. In *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, volume 2, pages 531–536, 2019.

[14] Vinh Quang Nguyen, Ton Hoang Nguyen, and Jae Wook Jeon. An adaptive fuzzy-pi clock servo based on ieee 1588 for improving time synchronization over ethernet networks. *IEEE Access*, 8:61370–61383, 2020.

[15] Henning Puttnies, Peter Danielis, Ali Rehan Sharif, and Dirk Timmermann. Estimators for time synchronization—survey, analysis, and outlook. *IoT*, 1(2):398–435, 2020.

[16] Henning Puttnies, Peter Danielis, and Dirk Timmermann. Ptp-lp: Using linear programming to increase the delay robustness of ieee 1588 ptp. In *2018 IEEE Global Communications Conference (GLOBECOM)*, pages 1–7, 2018.

[17] Kevin B. Stanton. Distributing deterministic, accurate time for tightly coordinated network and software applications: Ieee 802.1as, the tsn profile of ptp. *IEEE Communications Standards Magazine*, 2(2):34–40, 2018.

[18] Stefano Vitturi, Thilo Sauter, and Zhibo Pang. Real-time networks and protocols for factory automation and process control systems [scanning the issue]. *Proceedings of the IEEE*, 107(6):939–943, 2019.

[19] Gang Wang, Tianyu Zhang, Chuanyu Xue, Jiachen Wang, Mark Nixon, and Song Han. Time-sensitive networking for industrial automation: Challenges, opportunities, and directions, 2023.

[20] Martin Wollschlaeger, Thilo Sauter, and Juergen Jasperneite. The future of industrial communication: Automation networks in the era of the internet of things and industry 4.0. *IEEE Industrial Electronics Magazine*, 11(1):17–27, 2017.

[21] Xiong Xu, Zhenhua Xiong, Xinjun Sheng, Jianhua Wu, and Xiangyang Zhu. A new time synchronization method for reducing quantization error accumulation over real-time networks: Theory and experiments. *IEEE Transactions on Industrial Informatics*, 9(3):1659–1669, 2013.