EPiC Series in Computing

Volume 56, 2018, Pages 63–72

Proceedings of the 5th International OMNeT++ Community Summit

A Simulation Model of IEEE 802.1AS gPTP for Clock Synchronization in OMNeT++

Henning Puttnies[1], Peter Danielis[1], Enkhtuvshin Janchivnyambuu[1], and Dirk Timmermann[1]

University of Rostock, Institute of Applied Microelectronics and Computer Engineering, Germany
henning.puttnies@uni-rostock.de

Abstract

The aim of this paper is to describe a developed simulation model of the gPTP protocol for time synchronization in OMNeT++ using the INET library. gPTP is part of the IEEE TSN standards. Unfortunately, there is currently no simulation model of gPTP available. Therefore, we developed a new simulation model, compared it to results from the state-of-the-art, and would like to share it with the OMNeT++ community. The simulation model is based on the IEEE 802.1AS specification for full-duplex Ethernet according to a given network topology and use case scenario to analyze the results of the simulation as well as to provide a comparison with results from the state-of-the-art. Time synchronization and propagation delay measurements between time-aware systems are considered and results show that the simulation model works as expected.

1 Introduction

Real-time systems are systems that meet real-time requirements in task processing. Here, tasks must be completed within certain time intervals. Such tasks with real-time requirements are, for example, the data transmission in industrial networks or the processing of processes in industrial automation systems. The goal of real-time requirements is to process a task within a defined time interval triggered by an event or a predetermined schedule. The end of a task is also called a deadline. The tasks can occur periodically or aperiodically. In the case of periodic tasks, the events are repeated at regular intervals; in the case of aperiodic tasks, the event occurs irregularly and unpredictably. Real-time systems have soft and hard real-time requirements based on deadline compliance and the resulting consequences for the system. Missing a deadline in systems with soft real-time requirements only affects the performance, i.e., the quality of the results, but not the functionality. For systems with hard real-time requirements, this could have catastrophic and even safety-critical effects on the overall system. This could, for example, damage the system, or in the worst case, a person. For tasks in real-time networks to be able to meet their deadlines, the network nodes on which the tasks are running must be synchronized.

A. Förster, A. Udugama, A. Virdis and G. Nardini (eds.), OMNeT 2018 (EPiC Series in Computing, vol. 56), pp. 63–72

In this paper, we introduce a OMNeT++ simulation model[1] of the IEEE 802.11AS generalized Precision Time Protocol (gPTP) standard for synchronization specified by the Time-Sensitive Networking (TSN) task group [5]. TSN is an extension of standard Ethernet and adds real-time capability. We focus on Ethernet networks as wireless networks for real-time communication constitute a comprehensive research field of their own and wireless technologies such as 5G may be intended to complement but not replace wired technologies [2, 15]. Actually, Ethernet-based technologies tailored to the requirements in industrial environments have overtaken fieldbuses in terms of the number of newly installed nodes in factory automation [1]. The purpose of the simulation model proposed in this paper is to simulate the gPTP protocol in OMNeT++ using the INET library based on the IEEE 802.1AS specification for full-duplex Ethernet according to a given network topology and use case scenario to analyze the results of the simulation. At its current state, the simulation model considers the time synchronization and propagation delay measurements between time-aware systems. We want to note that the simulation model of the gPTP protocol is completely independent from the clock model. Therefore, we utilize only a simple clock model that assumes a constant drift in this paper.

The rest of this paper is organized as follows. Section 2 gives an overview of basics of gPTP. Section 3 describes the implementation of the simulation model. Section 4 presents simulation results. Related work is given in Section 5, and Section 6 concludes the paper.

2 Basics

In the case of Ethernet, gPTP uses a complete IEEE 1588-2008 profile, known as Precision Time Protocol (PTP) [4]. However, gPTP has some features that go beyond PTP. Compared to PTP, gPTP is much more robust against delay variations as it demands that every switch in the network supports gPTP at the MAC layer. Consequently, only an approx. constant propagation delay and no queuing delay can occur (c.f. [10, 11, 12]). The time synchronization of the standard is completely based on the master-slave principle. Each port of a time-aware system in a gPTP domain is in one of the states mentioned in the following.

- A master port is a port that sends time synchronization information to the slave port of a time-aware system located at the other end of the physical link.
- A slave port is a port that receives time synchronization information from the master port.
- A passive port is a port that is not gPTP capable.
- A disabled port is a port that is none of the ports mentioned above.

Furthermore, there are three different types of a time-aware system in a gPTP domain.

- The grandmaster (GM) exists only once in the domain. It initializes the clock synchronization periodically. It has one master port connected to the domain to send synchronization messages.
- There can be several bridges in the domain. Bridges comprise one slave port connected to the master port of another time-aware system and possibly several master ports depending on the network topology.

[1] The simulation model can be downloaded from https://gitlab.amd.e-technik.uni-rostock.de/peter.danielis/gptp-implementation

- In the domain, multiple slaves can exist, of which each has solely one slave port.

The GM of the gPTP domain is first selected using the best master clock algorithm (BMCA) defined in the PTP standard. However, the BMCA is out of scope in this paper as we are more interested into the synchronization precision. Furthermore, we prefer a static setup where we can configure which device should serve as GM. Before a time-aware system is then able to synchronize to the GM clock, it measures the propagation delay on each of its links, see subsection 2.1. Then time-aware systems finally exchange information to synchronize to the GM clock. The transport mechanism of the synchronization information between time-aware systems in described in subsection 2.2.

2.1 Propagation delay measurement

The propagation delay measurement is a media-dependent mechanism to measure the delay between a so-called delay requester and delay responder on each link connected to the ports of a time-aware system. If a port is gPTP capable, the delay measurement works as follows and is depicted in Figure 1(a) (adopted from [3]).

1. The delay requester, end station 1, sends a *Pdelay_req* message to the delay responder, bridge 1, at its local time t_1 and records t_1.
2. The *Pdelay_req* message is received by the delay responder, bridge 1, at its local time t_2. t_2 is recorded at the delay responder.
3. The delay responder responds at its local time t_3 by sending a *Pdelay_resp* message. t_3 is recorded at the delay responder.
4. The delay requester receives a *Pdelay_resp* message at its local time t_4. t_4 is recorded at the delay requester.
5. Then, the delay responder sends a *Pdelay_resp_follow_up* message containing the departure time t_3 of the *Pdelay_resp* message. t_3 is recorded at the delay requester.

Concerning the timestamp accuracy, note that it does not lead to inaccuracies that the simulation clock does not advance while frame processing. There is no processing delay in real-world gPTP systems, as gPTP timestamps the messages right after the start of frame delimiter (c.f. [5]). At the end of the message exchanges, the delay requester knows all four timestamps (t_1, t_2, t_3, and t_4) that it reads from the exchanged messages. The propagation delay is calculated using Eq. 1. The delay equals half of the difference of the intervals $(t_4 - t_1)$ and $r \cdot (t_3 - t_2)$. However, the local clocks have a frequency variation defined as clock drift, which results in different time values between distributed time-aware systems. Thus, both intervals must be based on a common time base and the delay responder interval $(t_3 - t_2)$ needs to be corrected by multiplying the interval by the rate ratio r of the delay requester to the delay responder clock frequency.

$$P_{delay} = \frac{(t_4 - t_1) - r \cdot (t_3 - t_2)}{2} \tag{1}$$

The equation above is based on the assumption that the link is symmetric since we consider only full-duplex Ethernet links. As stated in Eq. 1, the rate ratio r is the decisive factor for the delay measurement. In accordance with the standard, it is the ratio of the local clock frequency of the requester to the local clock frequency of the responder to determine the relative difference between two local clock frequencies as shown in the Eq. 2.

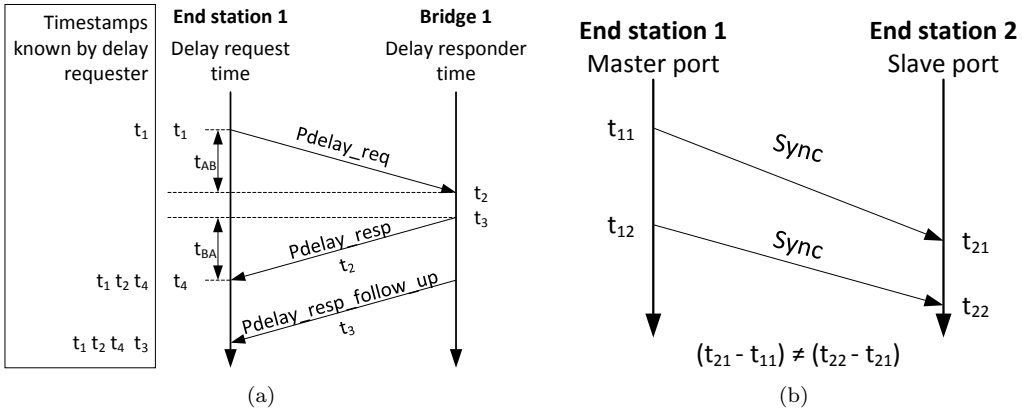


Figure 1: (a) Sequence diagram of the propagation delay measurement procedure (adopted from [3]), (b) impact of the clock drift on the arrival time of messages.

$$r = \frac{f_{requester}}{f_{responder}} \tag{2}$$

In our simulation, there is no model of the oscillator frequency. Hence, we need to approximate the rate ratio r without using the Eq. 2. As illustrated in Figure 1(b), two sequential messages arrive at the local time t_{21} and t_{22} of the slave port of end station 2. The time interval $(t_{22} - t_{21})$ between the arrival times of the messages is not equal to the time interval $(t_{12} - t_{11})$ between the departure times of the messages even if two messages are exactly of the same type and size due to the frequency inaccuracy of the local clocks of the end stations. However, the rate ratio r can be calculated using two intervals as shown in Eq. 3.

$$r = \frac{t_{12} - t_{11}}{t_{22} - t_{21}} \tag{3}$$

The main idea of Eq. 3 is to define the rate ratio using the timestamps of two successive sequential messages instead of the clock frequencies. Three conditions result from Eq. 3, t as described below.

- If the rate ratio r equals one, the local clocks of two time-aware systems work at exactly the same time.
- If it is less than one ($r < 1$), the clock of the end station 1 lags behind the clock of the end station 2.
- If it is greater than one ($r > 1$), the clock of the end station 1 precedes the clock of the end station 2.

The rate ratio r does however not tell about which the clock runs correctly. In our implementation, Eq. 3 has been used to calculate the rate ratio r. For more information about clock drift estimation, we refer to [6].

2.2 Transport of time synchronization information

The time synchronization in the gPTP domain is the same as the synchronization in case of a PTP boundary clock that uses the peer delay mechanism [5]. The time synchronization is

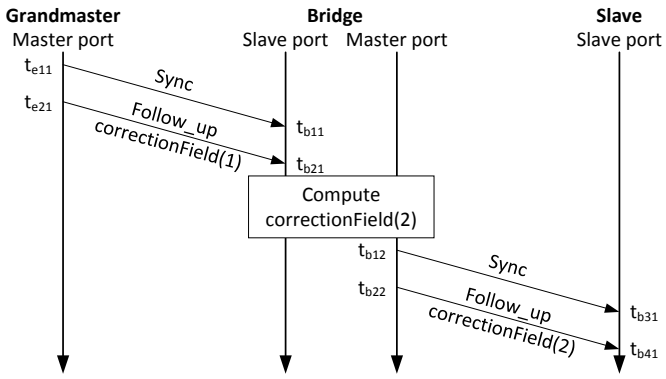


Figure 2: Transport of time synchronization information.

performed using *Sync* and *Follow_up* messages. End stations and bridges receive *Sync* and *Follow_up* messages on slave ports. For time-aware bridges, messages received on their slave ports are forwarded through each master port after the synchronization information has been corrected using propagation delay and residence time. Figure 2 shows the process over IEEE 802.3 full-duplex point-to-point links between three time-aware systems: two end stations (GM, slave) and one bridge. End station 1 sends a synchronization (*Sync*) message and timestamps the information at its master port at time t_{e11}. Bridge 1 receives the message and timestamps it at its slave port at time t_{b11}. After the *Sync* message has been sent on the master port to bridge 1, a *Follow_up* message is transmitted with following information at time t_{e21}.

- The *preciseOriginTimestamp* is the precise origin timestamp from the GM.
- The *correctionField* is the correction information updated by each time-aware system, which includes the propagation delay and the residence time of the system. For instance, in the case of the above scenario in Figure 2, bridge 1 computes a correction field as shown in Eq. 4 when a *Follow_up* message is sent. Then, the *Follow_up* message conveys the updated correction field to end station 2.
- The *rateRatio* is the frequency rate ratio relative to the GM clock.

$$correctionField(2) = correctionField(1) + P_{delay} + residenceTime + transmissionTime \quad (4)$$

The information carried by *Sync* and *Follow_up* is used to correct the local clock with the GM clock in each intermediate and end system. Finally, the clocks of all time-aware systems are synchronized to the GM clock using Eq. 5.

$$TimeSynced = preciseOriginTimestamp + P_{delay} + correctionField(1) + transmissionTime \quad (5)$$

The synchronized time equals the sum of the origin timestamp, propagation delay, correction field, and the transmission time between two end points of the physical link. The transmission time is calculated using Eq. 6. For our implementation, the propagation delay does not contain a transmission time. That is why the transmission time is added according to Eq. 5 and 6.

$$transmissionTime = \frac{packetSize\ [bit]}{dataRate\ [bps]} \quad (6)$$

3 Implementation

The main objective of the simulation model proposed in this paper is to implement core functionalities of gPTP in OMNeT++ 5.2 using the INET 3.6.3 library. Therefore, we defined the initial requirements of the project as follows.

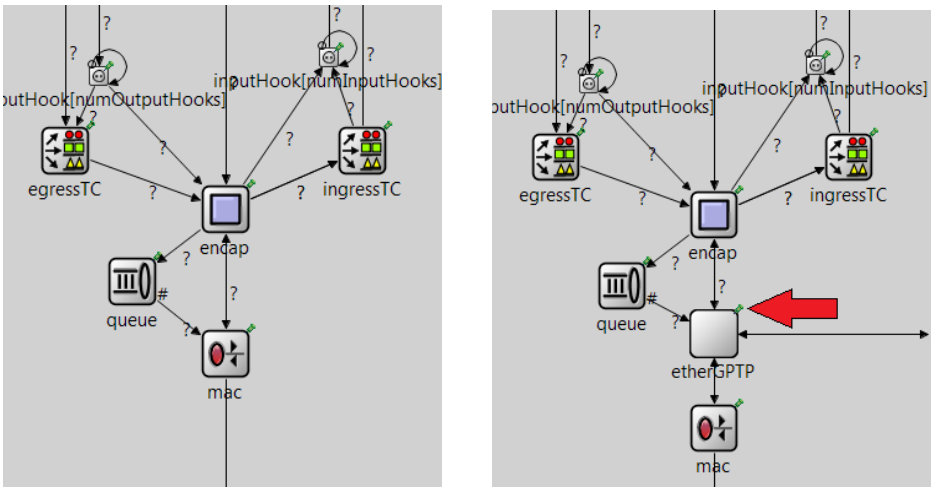
- The INET library should be used to integrate the implemented gPTP model seamlessly with other modules of networking standards or protocols that already exists in the INET.
- gPTP's best master clock algorithm is not considered for the simulation model. Thus, each port of time-aware systems has a pre-defined state as master or slave.
- Instead, we focus on the clock synchronization and propagation delay measurement.
- The implementation of a time synchronization protocol would be meaningless to simulate the network based on the time synchronization protocol without the introduction of basic clock drift. Thus, we have decided to integrate the basic idea of the clock with constant drift since we did not find a ready-to-use clock model that is supported by OMNeT++ 5.2 and INET 3.6.3.

3.1 Model of gPTP functionality

In order to satisfy the requirements stated above, we decided to develop a new simple module called *etherGPTP* that is located between the *encap* and *mac* modules within the *EthernetInterface* compound module of the INET library as highlighted by the red arrow in Figure 3(b). This *EthernetInterface* module that is depicted in Figure 3(a) is a model of a real physical network interface card. The *etherGPTP* module contains the main functionalities of gPTP, which are the time synchronization and propagation delay measurement. As implemented this way, our module can be integrated seamlessly with existing modules in the INET library. In accordance to the standard, gPTP needs to be implemented at the link layer of the OSI model to minimize the latency of the time synchronization. This was the main reason for implementing the new module *etherGPTP* in *EthernetInterface* that is located at the link layer of the INET library. A new compound module that includes the *etherGPTP* simple module is named *EthernetInterfaceGPTP* to differentiate it from the *EthernetInterface* module of the INET library. Each *etherGPTP* module has one of the following types: master or slave. These types need to be pre-defined for each port of a time-aware system before establishing the network.

- If its type is master, it sends a *Sync* message as requested by an upper layer module called *tableGPTP*, which is discussed in more detailed in the next section, as well as it receives a *Pdelay_req* message.
- If the type is slave, it receives a *Sync* message and synchronizes the local time based on the information carried by *Sync* and *Follow_up* messages. Also, it initiates a *Pdelay_req* message to measure the peer delay between two end points of the physical link.

The *etherGPTP* module receives messages from the lower layer module *mac* and then checks whether the message type is gPTP. If it is a gPTP message, the *etherGPTP* module processes it based on a type of the gPTP message, *Sync* or *Pdelay_req* etc. If the message type is not gPTP, the module forwards it to the upper layer without any modification of it. Messages from upper modules, *queue* or *encap*, are forwarded to the lower layer module *mac* since a gPTP message is only received from the lower layer module *mac* and the upper layer module *tableGPTP*.



(a) (b)

Figure 3: (a) The *EthernetInterface* compound module in INET, (b) the *EthernetInterfaceGPTP* compound module containing our *etherGPTP* module.

3.2 Model of gPTP messages

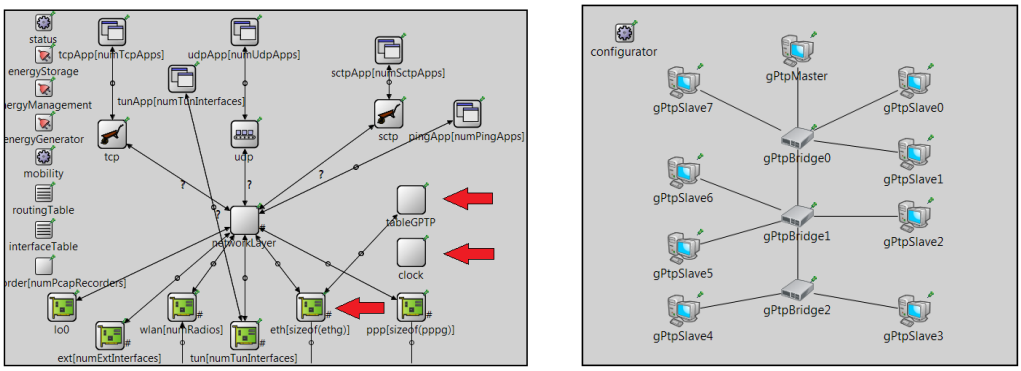
In compliance with the requirements, only time synchronization and propagation delay measurement related *Sync*, *Follow_up*, *Pdelay_req* and *Pdelay_resp* messages are implemented using the cPacket class of the OMNeT++ API. In INET 3.6.3, the etherType of the gPTP (0x88F7) is not known. To enable the INET library to identify the new etherType code, we would need to modify many existing modules, which requires much effort. Due to this reason, we decided to not implement the etherType of the gPTP. To exploit the existing possibilities of the library, gPTP messages are encapsulated within existing EthernetIIFrame in the INET library.

3.3 Clock model

In real-world scenarios, the inaccuracies of the crystal oscillator lead to clock drift. A clock does not run at the same speed as the reference clock due to several factors including temperature, proecess variation, and aging. We implemented a clock with constant drift for simplicity and each time-aware system has a distinct constant drift value. This model is not realistic as the clock frequency changes over time in reality. However, as the clock frequency change is relatively slow it can be assumed to be constant for relatively short simulation times.

3.4 Model of time-aware systems

In order to model time-aware systems, we use the existing *EtherSwitch* and *EtherHost* compound modules of INET. To enable them to recognize gPTP messages, the *EthernetInterface module* that is integrated in the *EtherSwitch* and *EtherHost* compound modules is replaced by the *EthernetInterfaceGPTP* module. Generally, time-aware systems can have multiple *EthernetInterfaceGPTP* modules (network interfaces). Those modules need to communicate with each other in order to forward a synchronization message to the master ports of the time-aware



(a) (b)

Figure 4: (a) Model structure of a time-aware system, (b) switched Ethernet-based network.

system. In order to accomplish this intercommunication, a new simple module, called *tableGPTP*, is introduced as mentioned above. The key responsibility of this module is to provide the intercommunication capability to the *EthernetInterfaceGPTP* modules that reside in the time-aware system. The slave port of the time-aware system receives a *Sync* message and synchronizes its local time. Afterwards the slave port requests the *tableGPTP* module that also resides in the same time-aware system to inform the master ports to send forward *Sync* messages. Each time-aware system has only one *tableGPTP* module. As illustrated in Figure 4(a), red arrows point out the newly implemented modules. Other submodules of the system are exactly the same as *EtherHost* and *EtherSwitch* modules.

4 Simulations Results

The goal of the simulation is to make sure that the newly implemented gPTP works as we expected and to analyze the results of the simulation against the results of a highly recognized research paper [8]. The propagation delay is measured to determine the accuracy of the algorithm and the time difference of all nodes to the GM after the synchronization.

4.1 Simulation setup

A network shown in Figure 4(b) presents the switched Ethernet-based network that consists of a master, three bridges, and eight slaves. It is taken from the research paper [8] because it is a realistic scenario for an in-car network. Each time-aware system's constant clock drift value is given as described in Table 1. A negative value expresses that the clock lags behind the simulation time of OMNeT++ whereas a positive value expresses that the clock precedes. The value zero means that the clock of the master node always runs at the simulation time.

Ma	Br0	Br1	Br2	Sl0	Sl1	Sl2	Sl3	Sl4	Sl5	Sl6	Sl7
0	30	-15	20	-50	10	50	-5	-50	40	-15	-35

Table 1: Clock drifts of the time-aware systems in ppm. Ma: Master, br: bridge, sl: slave.

4.2 Simulation Results

To calculate the propagation delay, Eq. 1 is applied. We set the propagation delay to 25 ms and the expected result is that delays converge to this value with small acceptable error. Actually, the error percentage is less than 1.8% and the absolute difference between real propagation delay and measured delay is less than 0.5 ns which is a very good result compared with the result of Lim et. al. [8] that accepts an error of ± 10 ns. Generally, the measured delay must be same as the real propagation delay since we have simulated the model, but there is a factor that causes an error for the measurement. Firstly, propagation delay measurement takes place every one second whereas synchronization message is sent from GM every 125 ms or 62.5 ms. Due to these different intervals, the clock drifts are different. The *Sync* message size is 44 bytes and a message size of the propagation delay measurement is 54 bytes. Because of different packet sizes, the packet transmission time is different. These time differences contribute to the error. Eventually, after synchronization the time difference of all nodes to the GM is zero, which shows that the gPTP simulation module works as expected.

5 Related Work

There are actually some implementations of PTP in OMNeT++. One of them is libPTP by Wallner et. al. [14]. However, it revealed to have a lot of dependencies and does not compile for OMNeT++ 5.2. Moverover, libPTP is only a PTP model whereas we propose a model of gPTP. Another solution is PTP++ [7]. This implementation represents a working simulation module, is easy to understand but is however not gPTP.

There are existing implementations of PTP (e.g., for the Linux operating system). However, integrating real world applications into OMNeT++ does not work seamlessly according to [9]. Moreover, it is hard to find a sufficient software implementation of gPTP, as it works on the MAC layer and is typically implemented in hardware.

Finally, to the best knowledge of the authors there is no official open-source implementation of gPTP for the latest version of OMNeT++. CoRE4INET represents an exception [13]. Our own tests however showed that it is quite complex and currently solely working for OMNeT++ 4.x.

Consequently, we presented an open-source implementation of gPTP for OMNeT++ 5.2 that can be used for simulating any networks using gPTP.

6 Conclusion

In this paper, we have proposed an open-source gPTP simulation model containing the main operations time synchronization and propagation delay measurements. The model uses clocks with constant drift and end stations as well as bridges have been modelled. Our implementation follows the gPTP standard and our simulation model works as expected. Moreover, in the scope of the project we have presented the results of the gPTP simulation model, which is built to analyze and verify the performance of the gPTP in the case of a switched Ethernet-based network. Our simulation model generates results that are comparable to other research papers such as [8]. We conclude that our model can be useful in simulating any networks using gPTP.

We concentrated on modelling the gPTP synchronization protocol and utilize only a simple clock model that assumes a constant drift in this paper. Integrating existing clock models, e.g. from [14], would be a reasonable extension for future work. Finally, a modification of the simulation model using the INET 4 dispatcher functionality might be considered in order to obtain an even more lightweight implementation of gPTP.

References

[1] Anybus News. Industrial Ethernet is now bigger than fieldbuses, 2018.

[2] Peter Danielis, Henning Puttnies, Eike Schweissguth, and Dirk Timmermann. Real-time capable internet technologies for wired communication in the industrial iot—a survey. In *In Proceedings of the 23rd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2018)*, 2018.

[3] Geoffrey M Garner and Hyunsurk Ryu. Synchronization of audio/video bridging networks using ieee 802.1 as. *IEEE Communications Magazine*, 49(2), 2011.

[4] IEEE Std 1588-2008. IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems, July 2008.

[5] IEEE Std 802.1AS-2011. IEEE Standard for Local and metropolitan area networks - Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks, February 2011.

[6] Kyeong Soo Kim. Asynchronous source clock frequency recovery through aperiodic packet streams. *IEEE Communications Letters*, 17(7):1455–1458, 2013.

[7] Martin Levesque. PTP++: Precision Time Protocol for INET, March 2015.

[8] Hyung-Taek Lim, Daniel Herrscher, Lars Völker, and Martin Johannes Waltl. Ieee 802.1 as time synchronization in a switched ethernet based in-car network. In *Vehicular Networking Conference (VNC), 2011 IEEE*, pages 147–154. IEEE, 2011.

[9] Christoph P Mayer and Thomas Gamer. Integrating real world applications into omnet++. *Institute of Telematics, University of Karlsruhe, Karlsruhe, Germany, Tech. Rep. TM-2008-2*, 2008.

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

[11] Henning Puttnies, Björn Konieczek, Jakob Heller, Dirk Timmermann, and Peter Danielis. Algorithmic approach to estimate variant software latencies for latency-sensitive networking. In *Information Technology, Electronics and Mobile Communication Conference (IEMCON), 2016 IEEE 7th Annual*, pages 1–7. IEEE, 2016.

[12] Henning Puttnies, Dirk Timmermann, and Peter Danielis. An approach for precise, scalable, and platform independent clock synchronization. In *Consumer Communications & Networking Conference (CCNC), 2017 14th IEEE Annual*, pages 461–466. IEEE, 2017.

[13] Till Steinbach, Hermand Dieumo Kenfack, Franz Korf, and Thomas C Schmidt. An extension of the omnet++ inet framework for simulating real-time ethernet with high accuracy. In *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques*, pages 375–382. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2011.

[14] Wolfgang Wallner. Simulation of the ieee 1588 precision time protocol in omnet++. *arXiv preprint arXiv:1609.06771*, 2016.

[15] 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.