This full text paper was peer-reviewed at the direction of IEEE Instrumentation and Measurement Society prior to the acceptance and publication.

A Simulation Framework for IEEE 1588

Wolfgang Wallner
Vienna University of Technology
Email: wolfgang-wallner@gmx.at

Armin Wasicek
University of California, Berkeley
Email: arminw@berkeley.edu

Radu Grosu
Vienna University of Technology
Email: radu.grosu@tuwien.ac.at

***Abstract*—IEEE 1588 specifies the Precision Time Protocol (PTP). The design space for PTP implementations is large, and system designers have to make trade-offs. A sophisticated and extensible simulation tool can assist PTP system designers when exploring the design space. This paper serves as an introduction to LibPTP, which is an OMNeT++-based simulation framework for PTP networks. LibPTP facilitates building PTP networks using Ordinary, Boundary, and Transparent clocks. For instance, it enables designers to study different configuration options, to compare delay mechanisms, or switch between clock servos. The paper gives an overview of the current feature set of LibPTP, and demonstrates LibPTP's capabilities through exemplary experiments. The demonstrations encompass analyzing the choice of synchronization intervals, the impact of path asymmetry, and daisy chaining of clocks.**

I. Introduction

Real-Time Systems (RTSs) are a special class of distributed systems that put an emphasis on the progression of real time in the distributed system. They establish a common notion of time among all participants in the distributed system. All participants are using local clocks to measure time. Similarly to other measurement systems, clocks are subject to measurement errors. Noise sources include the imperfection of the oscillator inside a clock as well as the imperfections of the communication system and processing inside the end nodes. RTS need to counter these noise effects to be able to provide a global time base of sufficient precision. Various synchronization techniques have been developed to establish such a global time base. One particularly important solution is the Precision Time Protocol (PTP), which is standardized in [1]. PTP is widely deployed in various industries and hardware support is becoming available in commercial off-the-shelf (COTS) devices.

When designing an RTS using PTP, it is of interest to justify the expected precision in advance, because PTP offers many configuration and implementation options. Simulation is a useful tool for design space exploration, to study design trade-offs, and to support design decisions with data. An easy-to-use, efficient, realistic and extendable simulation framework for PTP is therefore desirable when designing such systems.

This paper proposes LibPLN and LibPTP that are two libraries that facilitate the simulation of large-scale PTP networks:

- *LibPLN* is a portable, efficient software library for generating realistic oscillator noise
- *LibPTP* is a simulation framework including models for a PTP software stack, PTP capable Ethernet hardware, clock servos and other PTP-related components

LibPTP is based on the OMNeT++ simulation framework [9], and is an extension of the popular INET library [10] for OMNeT++. All software projects which have been developed for [2] are available under open-source licenses from the authors Github account[1].

The rest of this paper is structured as follows: Section II gives an overview on the requirements and related work. In sections III and IV we describe LibPLN and LibPTP, and how they can be used to simulate a PTP system in OMNeT++. Section V discusses exemplary experiments and demonstrates what kind of insights can be gained using the proposed simulation framework. Finally, section VII wraps up and draws a conclusion.

II. Simulating Clocks in Distributed RTS

This section introduces the main concepts with respect to clock synchronization in RTS and outlines related work.

A. IEEE 1588

The IEEE 1588 standard specifies the PTP, which is a protocol for network-based clock synchronization. PTP is designed to fill the gap between expensive high-precision synchronization mechanisms like the Global Positioning System (GPS) and other less expensive, but also less precise software-only solutions like for example the Network Time Protocol (NTP). Eidson gives a detailed introduction to PTP in [7].

B. Frequency Stability Analysis

When dealing with clock synchronization, we need to analyze and compare the frequency stability of individual clocks. The discipline that deals with these measures is referred to as Frequency Stability Analysis (FSA) [11]. This section gives a short overview of basic concepts and terms for FSA.

Allan Variance The Allan Variance (AVAR) is a two-sample variance that is commonly used to compare the frequency stability of different oscillators. While it has certain drawbacks that were improved by other variances, we prefer to use the AVAR here because of its widespread use in literature [11]. Also commonly found in literature is the Allan Deviation (ADEV), which is the square root of the AVAR.

[1] https://github.com/w-wallner/

978-1-4673-9615-8/16/$31.00 ©2016 IEEE

Time Deviation The instantaneous Time Deviation (TD) expresses how much the current time estimate of a given clock is ahead or behind the nominal, error-free clock.

Powerlaw Noise The disturbances that influence common oscillators are divided in two categories: deterministic influences (e.g. frequency drift, temperature sensitivity) and stochastic noise processes. Stochastic noise is commonly found in oscillators having a certain characteristic: the power spectral density of it's frequency fluctuations has the form $S_y(f) \propto f^{\alpha}$. Common values for α are the integers in the interval $-2..2$ [11]. Because of its characteristic power spectral density shape, this kind of noise is referred to as Powerlaw Noise (PLN).

C. Components in a PTP system

A PTP network consists of different components which act together, which we need to model for our simulation. The following presents the components that our PTP model comprises of as well as their current limitations.

- **Clocks:** Any real world clock suffers from noise, which is the reason why clocks in a distributed system need to be synchronized in the first place. The framework is currently limited to quartz oscillators, as they are the most common type of oscillator found in electronic devices. Moreover, we chose to focus on stochastic noise, and to disregard deterministic influences like temperature or pressure for now.
- **PTP software stack:** PTP is used to measure the offset of individual clocks in a network. The procedure how this is done is specified in IEEE 1588. For our simulation framework, we developed a standard compliant implementation that is suitable for execution in OMNeT++.
- **PTP capable hardware:** PTP can be implemented on different types of networks. Currently, the focus of the implementation is on Ethernet networks. As shown in other publications (e.g. [3]), hardware timestamping improves the reachable precision remarkably. For our simulation we have extended standard networking models from the INET library with PTP specific functionality, like the mentioned hardware timestamping.
- **Clock servo:** Once the synchronization error between a PTP slave clock and the master clock has been estimated, it can be corrected. A clock servo is a control algorithm that minimizes the synchronization error over time. Currently, the framework implements a PI controller for rate correction, together with logic for state correction in case the measured synchronization error is beyond a configured bound.

The described framework has been implemented by two software libraries, LibPLN and LibPTP, which can be used within OMNeT++. Both have been developed in the scope of [2], which also provides more in-depth information.

D. OMNeT++

OMNeT++ is a Discrete Event Simulation (DES) environment for network simulations. It was chosen as the basis for LibPTP, because it is easy to use, well documented, free for academic use, and has already been used in other publications on the simulation of PTP (see section II-F).

Simulation models in OMNeT++ can be combined and extended in an object-oriented way: their interfaces are defined in an OMNeT++-specific markup language, and their behavior is implemented in C++. Simple components can be combined to form more complicated components, and existing models can be extended. Another advantage of OMNeT++ is that it provides its users with support for all stages of simulation development, from initial model creation to carrying out parameter studies and finally plotting and analyzing the data.

This work uses the INET library, which implements OMNeT++ models for common network related components, like various software stacks or Ethernet switches. LibPTP extends standard network equipment models from INET with PTP functionality.

E. Clock model

When modeling clock synchronization, a very important model is that of the simulated clock device. The model we use is based on the simple clock model given in [8]. This model has of two components: an *oscillator* (or frequency standard) that ticks at a specified frequency, thus measuring the length of a given time interval, and a *counter* that counts the number of these ticks.

In a real device, each part of a clock could contribute to the overall noise. To keep our simulation model simple, we assume that the two basic clock components are perfect, and introduce a third, artificial component in between, i.e., a *Noise Generator*. The noise generator subsumes all noise effects in a clock. Hence, it is the only source of clock noise in our simulation model. A sketch of the complete clock model we use is shown in fig. 1.

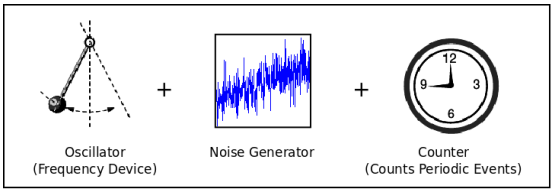

Figure 1. The clock model which is used in this paper. Both the oscillator and the counter are assumed to be perfect, and the noise is added by an artificial component, the Noise Generator. This image based on another image from [8].

As already stated, this work disregards the deterministic noise sources and focuses on the stochastic influences, that are implemented in LibPLN, described in section III.

F. Related work

1) Simulation of PTP: A basic simulation of PTP in OMNeT++ has been described by Steinhauser in [4]. Gaderer et al. have discussed the implementation of PTP and related topics such as oscillator noise in OMNeT++ in several papers. One of their publications ([5]) has served as an inspiration for the design of our PTP simulation framework.

2) Simulation of Oscillator Noise: The simulation of oscillator noise has been an ongoing research topic for decades, and it is well covered in the academic literature. The publications which were most relevant for the design and implementation of our clock noise simulation are those by Kasdin and Walter [6] and by Gaderer et al [5].

III. LibPLN

In [6], Kasdin and Walter propose a method for discrete simulation of PLN. A drawback of their method is that it is a batch method, and as such it is not really suitable for DES environments. This is also discussed by Gaderer et al [5], who claim to have successfully modified the Kasdin/Walter-method such that it fits the DES approach. The procedure described in [5] served as the baseline for the design of LibPLN.

Basically, LibPLN consists of mainly two parts:

- At its core, it implements the Kasdin/Walter algorithm using modern and portable math libraries (Boost C++ library and the FFTW C library).
- Around this core algorithm, LibPLN provides logic to produce PLNs at various frequencies, and to combine the individual samples to an overall noise sample. This combination is efficient. Unnecessary samples are left out and, thus, LibPLN saves computational complexity without influencing the overall result.

Using LibPLN it is possible to configure the characteristic noise of an oscillator that should be simulated. The oscillator model can then be sampled at arbitrary sampling rates, and the resulting noise samples will match the initial configuration. An example is shown in fig. 2: The black line in the background shows the configured ADEV, while the colored line in the foreground shows the simulated ADEV. The figure shows that, while the simulated sample does not match configured ADEV exactly, it is a good approximation over a large interval range.

The simulation speed achievable by LibPLN is roughly inverse to the simulated sampling rate, as shown in fig. 3. This property is important when the library is used in a DES environment, as we intend to do with our PTP simulations in OMNeT++.

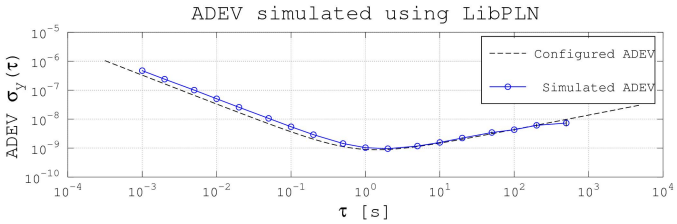

Figure 2. Example of an ADEV simulated with LibPLN. The black line in the background marks the configured ADEV.

A. Examples

By default, LibPLN ships with two pre-configured models:

- *AvgOsc*: An average oscillator as it could be found on a printed circuit board of common consumer electronic.

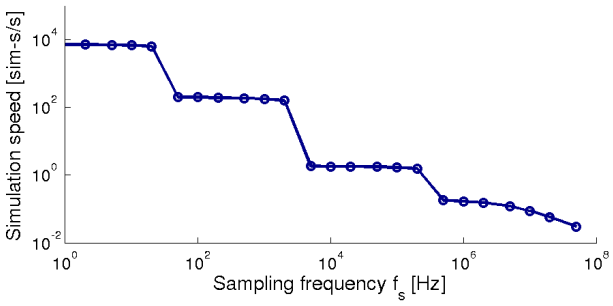

Figure 3. Simulation speed for different sampling rates, given in simulated seconds per real seconds. The steps are the result of optimizing for slower sampling rates.

- *WatchQuartz*: A model of quartz oscillator as it could be found in an off-the-shelf wrist watch.

The ADEV of these two clock models is shown in fig. 4. Additionally, the ADEV of a precision oscillator is shown to provide the reader with a reference. In section V these models demonstrate different PTP simulations.

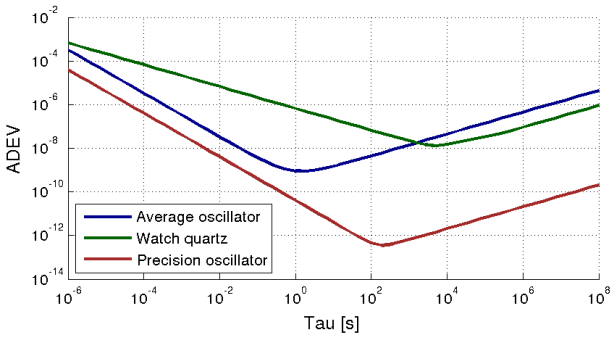

Figure 4. Side-by-side comparison of the ADEVs of the *AvgOsc* and *WatchQuartz* example oscillators. Additionally, the plot shows the ADEV of a precision oscillator as a reference.

IV. LibPTP

A. Overview

LibPTP is a collection of OMNeT++ models for simulating PTP-capable network nodes. It is an extension of the popular INET library, which provides base models for network-related components like routers or protocol stacks. LibPTP contains models of a PTP software stack, clocks, PTP-capable Network Interface Cards (NICs) and more. These individual parts can combined to simulate PTP-capable network nodes.

The current feature set of LibPTP for simulating PTP networks includes:

- All specified clock types: Ordinary Clocks (OCs), Boundary Clocks (BCs) and Transparent Clocks (TCs)
- Both delay mechanisms: End-to-End (E2E) as well as Peer-to-Peer (P2P) delay measurement
- Configuration options for all PTP relevant parameters, like e.g. `logSyncInterval`

- PTP/Ethernet as specified in Annex F of the IEEE 1588 standard
- Various configuration options for filtering the individual measurements
- A Proportional/Integral (PI)-based clock servo design, with the possibility to provide custom clock servos
- Limited support for various PTP profiles
- Sophisticated debugging and tracing options

While LibPTP provides models for many PTP options, it also has limitations. E.g. it currently does not support unicast communication, or features like signaling and management messages.

B. Example models

Convenience and easy-to-use were important design drivers for the simulation framework. To provide new users with a simple way to set up PTP networks, LibPTP comes with a set of preconfigured PTP network nodes that can be combined in the OMNeT++ Graphical User Interface (GUI) in a plug-and-play fashion. These example PTP nodes have icons that show their most important properties at a glance. Figure 5 depicts the encoding of these icons.

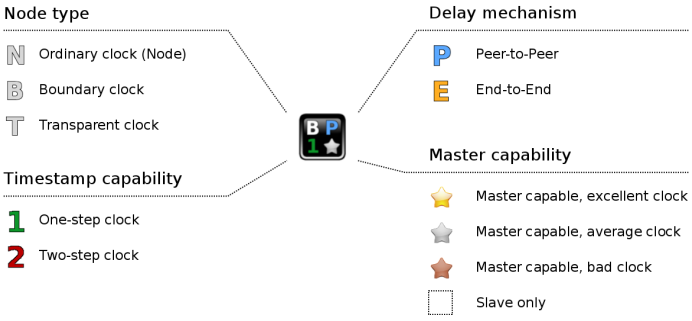

Figure 5. Encoding of the symbols used for the example nodes. The example shown in the center would be a one-step Boundary Clock, that supports P2P delay measurement and is master capable (with average clock attributes).

Figure 6 shows an example network using these node symbols. On the left side is a master-capable OC with a good clock that becomes the grand master of the network. It is connected to a BC, which is in turn connected to two daisy chains ending with OCs. The top chain consists of two slave-only TCs, while the bottom chain consists of a slave-only TCs and another BC. All nodes are one-step clocks, except the second TC in the top chain.

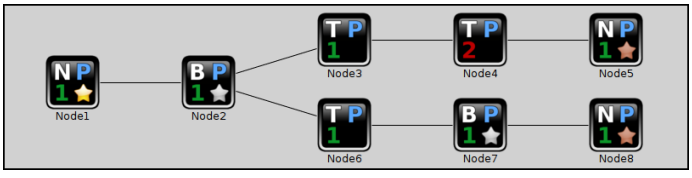

Figure 6. Example PTP network to show the individual node symbols.

V. Experiments

This section describes several experiments that demonstrate the capabilities of the simulation framework. The experiments should be representative to show the kind of insights can be gained using LibPLN and LibPTP.

A. Assumptions

The parameter space for a simulations is huge, so it is clearly not possible to cover all possible configurations. A detailed list of the assumptions that we used to constrain the simulation examples is given in [2].

B. Best Master Clock Algorithm

Validating our implementation of the Best Master Clock (BMC) algorithm (specified in [1]) is the first test item, before actually performing synchronization experiments. [Eid06] contains several theoretical test cases for the BMC algorithm. Case 5 from the book is the most interesting test case, because it is the most complex one.

A screenshot of our simulation of this test case is shown in fig. 7. The given network consists of OCs and BCs, where the top left node is the best clock in the network, and some nodes are wired in a ring. An execution of the BMC algorithm is expected to solve two challenges in this network:

- OC 1 should be elected as the grandmaster of the network, as it is the best available clock.
- The ring should be broken up: BC 8 is expected to move one of its ports which are adjacent to BC 5 and BC 7 to the `PASSIVE` state.

The tool-tip notification over BC 8 in fig. 7 is provided by LibPTP. The state description in the notification shows that the ports of BC 8 are indeed in the expected PTP states (The ports in the example are numbered counter-clock-wise starting from the top, and all BCs have 4 ports).

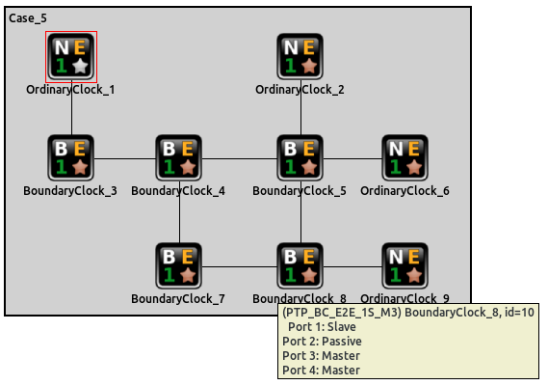

Figure 7. Simulation of the test case 5 network. The tool-tip in the screenshot shows the port states of BC 8. Port 2 is the port which is adjacent to BC 7.

Another important feature of LibPTP are its versatile debugging and tracing possibilities. For example, figs. 8 and 9 show the detailed traces of PTP port states of port 2 of BC 8 and its respective state decisions.

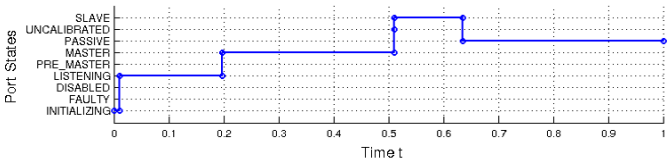

Figure 8. Port states of port 2 of BC 8.

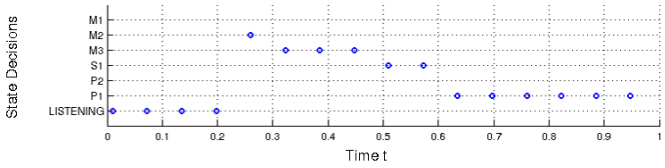

Figure 9. State decisions of port 2 of BC 8. The state decisions shown here refer to the definitions in the IEEE 1588 standard. The letter encodes the target state (*M* for master, *S* for slave and *P* for passive), and the numbers encode the reason for a given decision.

C. Sync Interval

One of the most important configuration parameters in a PTP network is the selection of the synchronization interval. This example simulates the effect of this parameter on a small network consisting of two nodes. The simulation is carried out two times with different oscillators for the slave node (*AvgOsc* and *WatchQuartz*, the two example oscillators of LibPLN). The clock of the master node is assumed to be perfect in both cases.

Resulting measurements for the mean and the jitter[2] of the slave's offset are shown in figs. 10 and 11. It can be seen in fig. 10 that the mean value of the slave's offset decreases for smaller sync intervals, until it converges to the same value in both cases. Figure 11 shows a similar behavior for the jitter of the slaves offset, but in the case of the *AvgOsc* example it converges to a much lower value than for the *WatchQuartz*.

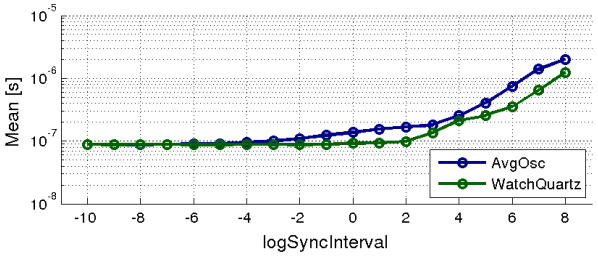

Figure 10. Mean value of the measured offset from the master clock.

D. Path Asymmetry

The offset estimation specified by PTP is based on the assumption that the path delay between two nodes is identical in both directions. As this assumption does not always hold in real networks, section 11.6 of [1] specifies *path asymmetry correction*. Using LibPTP, we can visualize the effects of

[2]The term *jitter* is used to refer to the peak-peak difference.

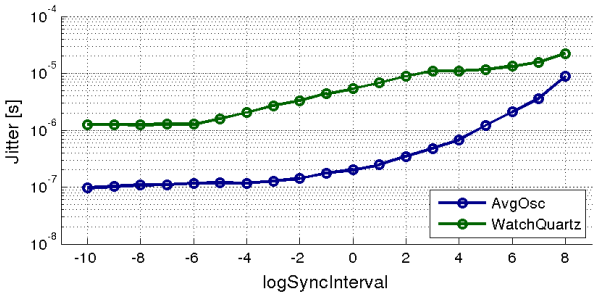

Figure 11. Jitter of the measured offset from the master clock.

path asymmetry on a slave's offset and how path asymmetry correction can be used as a counter measure. For this purpose, we simulate a network consisting of a master and a slave node three times with different configurations:

1) with completely symmetric paths
2) with asymmetric delays in the PHYs of each node, but without any asymmetry correction
3) with asymmetric delays as in the previous case, but this time with asymmetry correction configured in the slave

Figure 12 shows the `offsetFromMaster` that the slave estimates in each of the three cases. The actual offset from the slave to the master is shown in fig. 13. It can be seen that the slave thinks that its offset is quite low in all the simulated cases, as it cannot measure the path asymmetry. But as fig. 13 clearly shows, this is not true in the case 2, when the path is asymmetric but not asymmetry correction is configured. On the other hand, configuring the correct asymmetry correction leads to similar good synchronization results like in the first case.

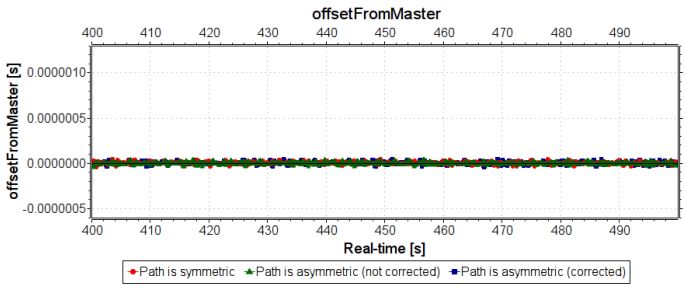

Figure 12. Measurements of the `offsetFromMaster` value.

E. Daisy Chain Configurations

As the final experiment for this paper, we carry out a similar parameter study for synchronization interval as in section V-C. But this time on a larger network, consisting of an Ordinary Clock acting as the grandmaster, and 50 PTP nodes connected to it in a daisy chain. These nodes are assumed to be either Boundary Clocks or Transparent Clocks. We then measure the maximum Jitter of every tenth node.

Figure 14 shows the result . As one would expect, it holds that a larger distance to the master leads to an increased jitter for all configurations.

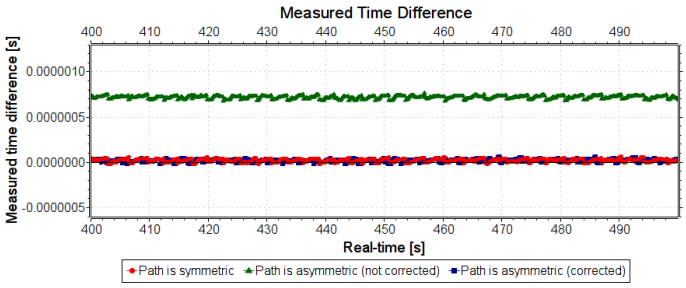

Figure 13. Measured clock differences between the master and the slave node.

For longer sync intervals the comparison of BCs and TCs also behave as expected: while the jitter increases for both clock types with longer sync intervals, TCs perform much better than BCs. A surprising result is the jitter for shorter sync interval values: At `logSyncInterval` $= -2$, there is hardly any difference between these two types of clocks, and for faster sync intervals BCs actually performed *slightly better* in this experiment!

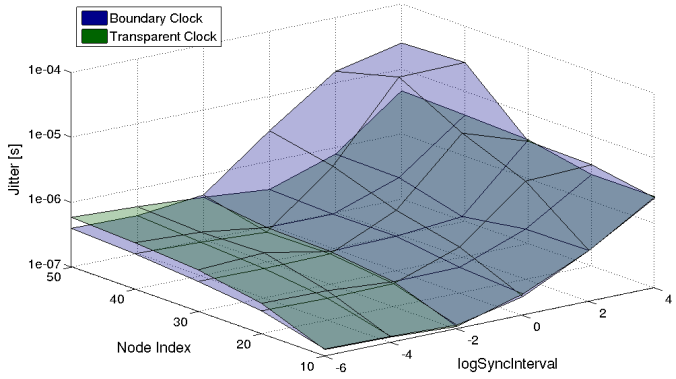

Figure 14. Jitter comparison for daisy chains of Boundary Clocks and Transparent Clocks for different sync intervals.

VI. Discussion

The results of our PTP simulation evaluation can be summarized as follows:

- It is feasible to modify the batch approach for Powerlaw Noise simulation proposed by Kasdin and Walter to fit the requirements of a Discrete Event Simulation environment.
- Users of LibPLN can use define arbitrary Powerlaw noise attributes, and thus simulate their oscillator of choice.
- The features implemented in LibPTP already provide the user with an easy to use way for the exploration of large parts of the PTP design space.
- The generic architecture LibPTP provides interfaces for custom extensions, like for example user defined clock servo algorithms.

While the combination of LibPLN and LibPTP can already be used to gain insight into the behavior of PTP networks, the current state of our simulation framework only solves a part of the overall problem. Examples for desirable future improvements include the following:

- **Extending PTP options:** LibPTP implements a subset of the possible PTP options. Interesting extensions would be of unicast support, UDP as the transport layer as well as Management and Signaling messages.
- **Mainlining LibPTP:** Currently the LibPTP library depends on the INET library, but is developed independently. If the internal structure of the referenced INET models changes, these two libraries could become incompatible. Thus it would be useful to try to mainline at least parts of the functionality back to INET.
- **Modeling additional noise sources:** The current simulation model completely ignores noise sources such as temperature or pressure. To gain more realistic simulation results, it would of course be interesting to add these influences to the simulation model.

VII. Conclusion

LibPLN and LibPTP are two main components to build a simulation framework for PTP. The experiments conducted with the scope of this research have proven that both LibPLN and LibPTP provide a way to gain insight in the domains of clock noise and time synchronization. To the best of our knowledge, there were no free software components available to efficiently and realistically simulate oscillator noise as well as PTP networks other than [2]. With this work, we hope to contribute to the further distribution of the IEEE 1588 protocol and enable users to efficiently design and provision PTP networks.

Acknowledgment

This research has received funding from the ARTEMIS Joint Undertaking (JU) under grant agreement no 621429.

References

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

[2] W. Wallner, *Simulation of Time-synchronized Networks using IEEE 1588-2008* Master's thesis, Faculty of Informatics, Vienna University of Technology, 2016.

[3] Y. Li, *A Study of Precision of Hardware Time Stamping Packet Traces* Proceedings of the 2014 International IEEE Symposium on Precision Clock Synchronization for Measurement, Control and Communication, Austin, 2014.

[4] J. Steinhauser, *A PTP Implementation in OMNET++* Bachlors thesis, Faculty of Informatics, Vienna University of Technology, 2012.

[5] G. Gaderer et al, *Achieving a Realistic Notion of Time in Discrete Event Simulation* International Journal of Distributed Sensor Networks, 2011.

[6] N. Kasdin and T. Walter, *Discrete Simulation of Power Law noise* Proceedings of the 1992 IEEE Frequency Control Symposium, 1992.

[7] John C. Eidson, *Measurement, Control, and Communication Using IEEE 1588* Springer, 2006.

[8] D. W. Allan et al, *Application Note 1289: The Science of Timekeeping* Agilent Technologies, 2000.

[9] *OMNeT++ Discrete Event Simulator* [Online] Available: https://omnetpp.org/

[10] *INET Framework* [Online] Available: https://inet.omnetpp.org/

[11] W. J. Riley, *NIST Special Publication 1065: Handbook of Frequency Stability Analysis* National Institute of Standards and Technology, U.S. Department of Commerce, 2008.