Implementation and Performance Analysis of Precision Time Protocol on Linux based System-On-Chip Platform

Mudassar Ahmed, Robert Manzke
Faculty of Computer Science and Electrical Engineering
Kiel University of Applied Sciences
Kiel, Germany

***Abstract*—Precision Time Protocol (PTP) claims to overcome time synchronisation challenges; it provides a self-organising time synchronisation protocol while utilising the existing ethernet based network capabilities for timekeeping applications. Time the message is sent or received at master/slave device is recorded (Timestamped), the precision of this time measurements is essential, it sets a fundamental level of accuracy for time synchronisation. The Timestamp for a PTP message is generated either on Software or Hardware level. A research objective of the study is to analyse the precision uncertainties and maximum attainable accuracy in both hardware and software timestamping-based implementation of the protocol. As the PTP protocol does not require additional physical network infrastructure to establish a primary PTP network. So, to analyse the possible effect on the performance of established PTP network, the simulated load is applied in different dimensions (CPU, I/O and Network) on PTP nodes during clock communication. The paper presents the summary of the PTP protocol, PTP infrastructure in Linux, and it presents the results collected during the different test case scenario, the results are further analysed and evaluated to draw a conclusion.**

***Index Terms*—Precision Time Protocol, PTP, IEEE 1588, Time Synchronisation, Timestamping**

I. Introduction

Continuous transformation of computer systems from large-scale isolated units to application-specific distributed units is rising the challenge of time synchronisation due to their associated and time-critical actions. The specific clock in a networked system needs to be checked whether the time deviation is acceptable for the particular application or there is a need for correction in time. As the distributed systems are becoming more complex and modular, where each module/node is communicating via a standardised communication medium, the demand for precision and accuracy of time synchronisation is increasing, IEEE 1588 Precision Time Protocol (PTP) developed to overcome these challenges.

The widespread adaptation of Precision Time Protocol (PTP) is not just limited to industrial automation and measurement systems, the integration of PTP is also realised and standardised in many leading technologies/sectors like Audio-Video Bridging, Smart Power Grids and Financial Systems. Due to widespread recognition of PTP, the silicon vendors are also offering an onboard sense of time support with hardware-assisted timestamping capabilities for the wide variety of solutions. Additionally, official integration of hardware timestamping and PTP Hardware Clock (PHC) API in mainline Linux kernel is becoming a catalyst in the development and adaptation of Linux based software solution for time synchronisation applications in different sectors. It is probable that different Linux kernel resource-scheduling procedures can affect the time timestamping process. The source of these latencies can be other resource exhausting applications running on the same system. In some cases, these resource-exhausting applications cannot be avoided as an overall system.

A. Research Objectives and Goals

Goals:

- Enabling Hardware timestamping capabilities of Linux bases SoC Platform (BeagleBone Black).
- Analyzing the behaviour of open source software solution for PTP implementation while simulating different load scenarios (Network load, Processing load) in Linux OS.

Research Objectives/Questions:

- Analysis of precision uncertainty in Hardware and Software-based solutions of PTP.
- What is the maximum attainable accuracy with Hardware and Software-based solutions of PTP?

B. Approach and Outline

The protocol hierarchy is established using multiple IEEE 1588 clock based SoC (hardware-assisted) platform in different test case scenarios (Hardware/Software Assisted and with Simulated load), an open-source solution [3] is used to establish the network in Linux environment. The log data generated in a specific scenario is prepared and imported to MATLAB workspace, where comparison of results in different scenarios are compiled in graphical form.

First, the paper introduces to the time synchronisation and give an overview of the PTP protocol in section **II**. Section **III** focuses on the Linux support for PTP in the context of the timestamping mechanism and Clock control mechanism. Implementation aspects are discussed in **IV**. The results of the test scenarios are presented and discussed in Section **V**. Finally, the Section **VI** draws the final conclusion on work.

II. Literature Review

A. Time Synchronization

The time is a measurable period in which a particular event, process or action may occur. In the domain of distributed systems, it can be used to conclude the rate of change, event ordering, an interval between different process and exact time of particular action. A clock is used to measure the specific spot in the time or a whole interval of time. In an embedded system, the clock is driven by an internal oscillator, which generates the signals with a precise frequency. In a networked system, where different nodes are having diverse type of clocks, which are powered by non-identical oscillators. These oscillators are running at different frequencies and having different behaviour in different conditions (Temperature, Voltage) [9], which results in a timing error. So, there is a need for time synchronisation to correct the drifting clocks from the reference time. There are several methods/principles in time Synchronization. [8] [7]

External Synchronization: In External Synchronization all nodes are synchronised with external time source, that can be via the Internet using NTP or using GPS.

Internal Synchronization: In Internal Synchronization nodes are synchronised with each other via establishing a master-slave hierarchy base network or with a reference clock. It is not necessary to synchronise with an external time source.

Hybrid Synchronization: This method is similar to Internal Synchronization, but the reference/master clock is tuned with an external time source.

TABLE I
Comparison of Time Synchronization Technologies [10]

	GPS	NTP	IEEE-1588
Spatial Extent	Wide Area	Wide Area	A few Sub-nets
Communications	Satellite	Internet	Network
Target Accuracy	Sub-Microseconds	Few Milliseconds	Sub-Microseconds
Hierarchy	Client/Server	Distributed	Master/Slave
Administration	N/A	Configured	Self-Organizing
Hardware Assisted	YES	NO	Optional (For High accuracy)

NTP: The main focus of the NTP protocol is on the systems, which are spread over a wide network, the target device is synchronised with a time server over the internet.

GPS: The GPS systems communicate via satellite, which provides different services including timing. Regarding time synchronisation, GPS is used for autonomous systems which are remotely located in the certain area and receiving timing information via satellite communication.

B. Overview of IEEE 1588 Precision Time Protocol (PTP)

Precision Time Protocol was first introduced in 2002 [1] and then further extended in 2008 [2]. PTP hierarchy is based on master/slave relation of clocks, the hierarchy is established using PTP enabled devices, which are mainly real-time clocks. In the protocol hierarchy, the top-level clock is called Grandmaster-Clock, which is treated as a reference time in the network, and other clocks can perform a role of slave or master. The Best Master Clock Algorithm (BMCA) ensures the self-organising property of protocol, by enabling PTP device to choose a specific role in the hierarchy, either in any case of failure of a master clock or in obtaining a master clock role on the base of clock data sets.

C. Protocol Standard Messages

The whole hierarchy of PTP system is based on the PTP messages exchange, which enables every PTP device from the bottom level of the hierarchy to synchronise with their master device, this chain reaches to grandmaster clock/reference clock. PTP messages are categorized in General Messages and Event Messages.

Event Messages: Event messages are timestamped messages, which are critical for accuracy in delay calculation mechanism of PTP.

General Messages: General messages are ordinary data transmitting or configuration messages, which are not timestamped but used to transport the timestamping information.

D. Protocol Standard Devices

Ordinary Clock (OC): The OC is a single port clock, which can be a master or slave clock. It communicates with other PTP devices through a single communication path. [1]

Boundary Clock (BC): The BC node has more than one ports to communicate within the network through multiple communication paths. It is used to further extended the network and each port of BC act as an OC. In the case of failure of reference clock, it will act as a time source in the network. [11]

Transparent Clock: There is no master/slave state in Transparent clock; it has multiple communication ports for transferring PTP messages. During forwarding messages from input port to output port, it computes delay caused by a device in transferring process of the PTP messages. The computed delay is called residence time, this residence time then added to designated correction field part of particular timing message. [11] [2] [12]

E. Message Exchange and Delay Computation

In the synchronisation process, the time information is exchanged between master and slave for the calculation of offset between clocks and delay caused by the network infrastructure. For that, there are two mechanisms defined, which are used with different combination of network infrastructure. Overview of the mechanisms is given below.

1) Delay Request-Response Mechanism: The mechanism yields mean path delay (Average of the time taken by data to travel between slave and master). As shown in Figure 1, *Sync message* is timestamped at t_1 and sent to the slave; practically the *Sync message* is sent at t_{2m} due to unknown network delays. So, another *Follow_Up message* containing the timestamp value of t_1 is sent to acknowledge the slave about the specific instance, when the *Sync message* was timestamped. Similarly, *Delay_Req message* is timestamped at t_3 and received at the master at t_4 instead of t_{3m} due to delays, the master clock sends a response message, which contains the exact time when *Delay_Req message* was received.

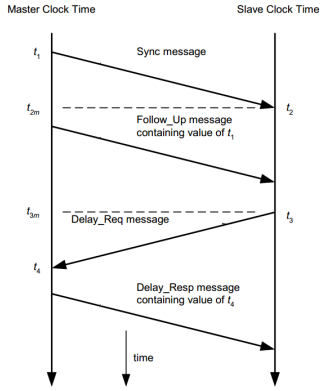

Fig. 1. PTP Delay Request-Response Mechanism (End-to-End) [6]

$$MeanPathDelay = \frac{(t_2 - t_1) + (t_4 - t_3)}{2} \quad (1)$$

$$OffsetFromMaster = (t_2 - t_1) - MeanPathDelay \quad (2)$$

2) Peer Delay Mechanism: In this mechanism of delay calculation is same as the end-to-end mechanism, but instead of *Delay_Req message*, the *Pdelay_Req*, and *Pdelay_resp* messages are exchanged, these messages are sent to the port which is immediately connected. [2] [12]

F. Self-Organising Hierarchy Establishment Mechanism

As a self-organising protocol, master/slave states of the clock are determined by the best master clock algorithm (BMCA), every ordinary (OC) and Boundary clock (BC) is equipped with BMCA. The algorithm enables OC/BC to determine their states locally instead of negotiating within a network to decide the clock state. [7] For every clock, there are standard clock properties (Priority1, Class, Accuracy, Variance, Priority 2 and Unique Identifier) are defined, which are used by the BMCA to compute the state of clock [2]. These properties can be used to manipulate the behaviour of BMCA in order to assign the particular role to a specific clock. A clock, either explicitly configured (clock properties) as the best clock or consider itself on the bases of BMCA as an eligible master clock, in both cases the clock advertises its clock properties via sending announce message in the network. In case of failure of the master clock or recognition of another better clock in the network, then the master-state associated messages (Sync and Announce) stops, and another clock takes the role of the best master clock. [13]

III. PTP Infrastructure in Linux

The journey of time synchronisation begins with NTP (Network Time Protocol), and David Mills is known as the father of NTP. Many of Mills purposed method for timekeeping and synchronisation of internal and external clocks can be seen in more advanced and accurate time synchronisation protocol like PTP. [4] The first version of PTP was standardized in 2002 and three years later Kendall Correll introduced a first opensource software-only solution (*ptpd*) [5] for implementation of PTP on Linux based systems. The solution became a base for further development in this domain. Later in 2009, Patrick Ohly introduced hardware timestamping in Linux kernel [3], Ohly then altered existing version of ptpd and extend his support for hardware supported synchronization [6]. Although hardware timestamping mechanism was already introduced in the Linux kernel, but there was no proper solution to control the hardware clock. So, in 2010, Richard Cochran introduced PTP clock infrastructure in Linux [3]. Later in 2011, Cochran presented the LinuxPTP tool for hardware and software based time synchronisation solution using hardware timestamping and PHC API of Linux kernel [4]. These opensource solutions and Linux kernel support are playing a vital role in the development and widespread use of this protocol. Many developer and researchers are using and extending these tools to realise their concepts on different platforms for intended applications.

A. Timestamping Mechanisms

Software Timestamping: Timestamps can be generated at different layers of a network; software timestamping is most widely and readily available option. In software timestamping, SO_TIMESTAMP options of Linux kernel is used which is the most widely exercised options in opensource implementations of PTP protocol. As shown in figure 2, the packets are timestamped at userspace using system time. Furthermore, timestamped packet still need to pass through other layers of the network in order to reach the physical channel of the network, which also causes some amount of error.

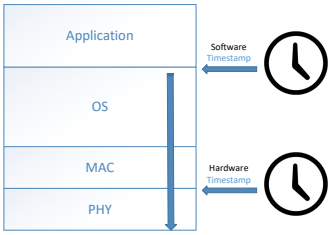

Fig. 2. Software and Hardware Timestamping

Hardware Timestamping: Hardware-based timestamping effectively reduces the jitter caused by OS level uncertainties and reduces the error resulted by network layers. Hardware timestamping can be either on *MAC* or *PHY* layer, it depends on the type of hardware. Linux network stack supports hardware timestamping by using SO_TIMESTAMPING feature.

B. PTP Clock Infrastructure and Control API

The first open source PTP solution (ptpd) was released in May 2005, which supports software only PTP implementation. Although many attempts are made to integrate PTP hardware support by using Linux *ioct* procedures and adding many preprocessor commands in the software-only solution of Kendall Corrells, which made existing code more unsuitable for support of multiple hardware [3]. After the official integration of hardware timestamping, which was proposed by Patrick Ohly,

there was no PHC clock infrastructure in the Linux although Ohly proposed two concepts (Assisted system Time and Two-level PTP) for the synchronization of system clock with PHC in his publication [6] but there was no efficient mechanism was proposed to control the PHC clock [3]. In order to overcome these problems, In 2010 Richard Cochran introduce a PTP clock infrastructure, which was then integrated to Kernel version: 3.0. The solution includes the PTP hardware clock (PHC) drivers architecture and a standard mechanism in the form of API to control PHC.

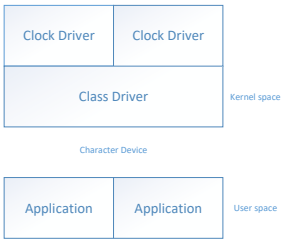

Fig. 3. PTP Infrastructure in Linux [3]

Figure 4 shows the clock infrastructure, where the class drivers cover the more generalised features like creation of character device, validation of *ioct* calls and management of the timestamped event ordering, and specific clock needs to provide the clock driver which covers the hardware aspects of the specific clock. The specific clock has to register their clock driver with the class driver which will generate the Character device; the Character device will be accessible to userspace via PHC userspace API. The approach which is used to control the PHC is quite similar to NTP timer model. [3]

IV. Design and Implementation

According to application requirements, the PTP network can be established with hardware assistance or pure software based. In order to implement the PTP protocol on Linux based SoC platform, we chose LinuxPTP, a most reliable opensource PTP implementation for UNIX like operating systems. Although there is another solution (*PTPd*) available which also claims hardware timestamping and PHC control based implementation, due to lack of compatibility for multiple hardware, we preferred to use linuxPTP which is compatible for chosen hardware platform and easily extendable to implement the additional features. Regarding hardware platform, *Beaglebone Black* is selected due to its support for hardware timestamping for PTP protocol and wide acceptance in the open source community as reference implementation platform. Additionally, In order to utilize this hardware assistance, *CONFIG_PPS* and *PTP_1588_CLOCK* Kernel option should be enabled. Furthermore, for performance analysis of PTP protocol on the selected hardware-software combination, *stress-ng* and *iperf* tools are used to put some stress on reference implementation platform in different dimensions (CPU, Network). Finally, the data collected from tests is cleaned and imported to *MATLAB* workspace, where the data is compiled to graphical form.

A. Tools and technologies

1) LinuxPTP : LinuxPTP was introduced by Richard Cochran [4] in 2011 after integration of PHC API in Linux mainline kernel. There was two synchronisation mechanism purposed by Patrick Ohly [6] to synchronise the system and PHC clock. The LinuxPTP solution closely resembles the second method which is "Two-Level PTP" [4]. The LinuxPTP project yields multiple executable files in order to establish the whole two-step synchronisation mechanism.

ptp4l: The tool synchronizes (by default) PHC clock with master clock in network. If the system does not have PHC then it synchronises the system clock using software timestamping.

phy2sys: The tool synchronizes Linux system clock to the PHC. In whole process the the *phc2sys* is synchronized with *ptp4l*, where system clock act as slave and PHC plays a role of master clock.

pmc: *pmc* is the realisation of PTP management client as defined in the standard. The tool is used to get the extra information from the network like identity, path delay and accuracy.

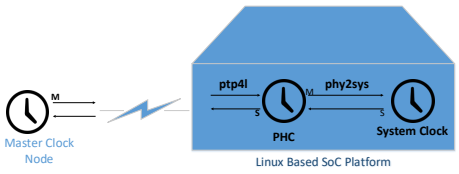

Fig. 4. LinuxPTP based Hardware Assisted Time Synchronization

B. Design Consideration and System Hierarchy

Flexibility is one of the forefront features of the PTP protocol, and it is optional to use all PTP supported network components, the network components include routers/ switch. Although the hardware platform provides the hardware support but the used network router (TP-Link TL-WR940N N450) does not provide PTP support. So, the maximum/ dependable accuracy and the precision is not guaranteed. On the other hand, only the end-to-end delay calculation mechanism is exercised. The fig 5 shows the architecture of the test environment.

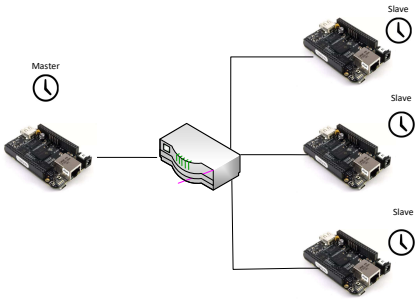

Fig. 5. Architecture of Test Environment

V. Test and Measurements

Regarding test case scenarios, first, pure software timestamping-based network is established and results are collected through standard logging. In next section the results from pure hardware timestamping-based solution are collected, after that, a comparison between the software and hardware-based solution is drawn in the context of offset from master clock and precision of synchronisation process. In the following sections, there are some CPU, I/O and network-based stress tests are conducted on hardware-assisted time synchronisation implementation. In order to present the results in a coherent form, data fitting process with default Smoothing Splines model from Matlab tool is used. In some graphs, we found possible and necessary to present the raw data points parallelly. So, where ever the data is shown in the description window of the graph, it shows the raw data of the respective slave.

1) ***Software Timestamping****:* In the software-based scenario, network hierarchy is established using one Beaglebone device acting as a master clock and three Beaglebone devices acting as slave clocks in LAN based network.

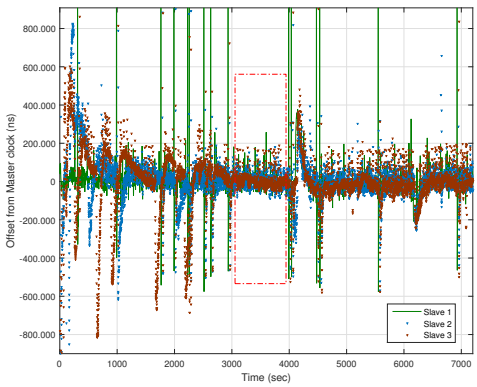

Fig. 6. Software Timestamping based Time Synchronization I

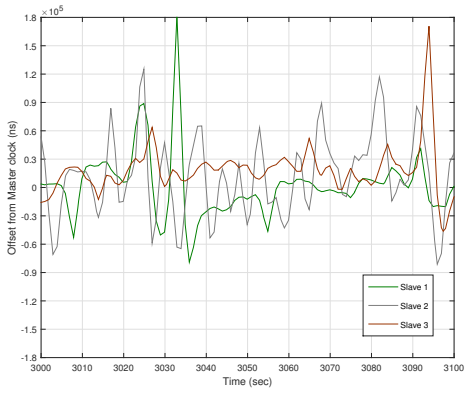

Fig. 7. Software Timestamping based Time Synchronization II

Figure 6 shows the offsets of slave clocks over the span of approximately two hours, where the drift of slave clocks are up to 2ms, and it can also be seen that clocks are showing uncertain behaviour. Overall the offsets are between $\pm$ 0.5ms, but due to an unknown error, there are sudden changes in offset in all clocks, which may be the result of a change in speed of the clock. If we take a closer look at the relatively stable part of test duration, which is highlighted by a rectangular shape in figure 6. Figure 7 is drawn on the basis of data taken from stable synchronisation span of figure 6. In this graph, only 100 samples of data are considered, in order to visualise the results more clearly. During whole span (approx 15 min) of synchronisation, the offset remained under 300 microseconds.

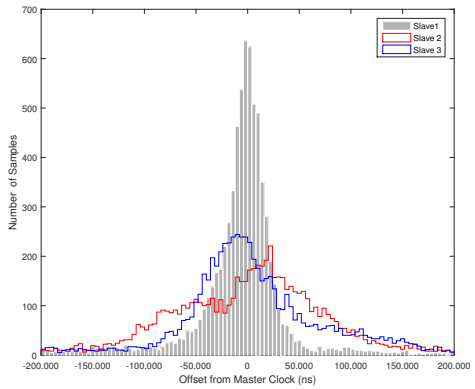

Fig. 8. Precision of Software based PTP Implementations

Figure 8 shows the precision of clocks, where the number of samples by specific clock are plotted between the range of -200us to 200us. It can be seen that slave 1 have relatively better precision as compare to other slaves clocks, where the offset fluctuation can clearly be seen.

2) ***Hardware Timestamping****:* The figure 9 shows the synchronisation of slave clocks over the period of twelve hours. The synchronisation period is divided into two parts. In the first part, the synchronisation network consists of three slaves and one master clock, performing their roles for approximately 1.3 hours and in the second part of synchronisation period, two slaves are removed from the network in order to analyse the behaviour of synchronisation process with fewer slaves communicating with the master clock. It can be seen that after removing slaves form network, results form remaining slaves are relatively stable and also remained stable for a long period.

The figure 10 shows the magnified version of first half of the previous figure where three slaves are active in PTP network and continuously synchronising their time with master clock by every second and the accuracy of slave clocks is between $\pm$ 250ns. Figure 11 reflects the second half of figure 9, where only one slave is correcting their offset with one master clock, and the other two slaves are removed from the network. It can be clearly seen that the results are remarkable with fewer slave devices in a network. The offset is fluctuating between the -

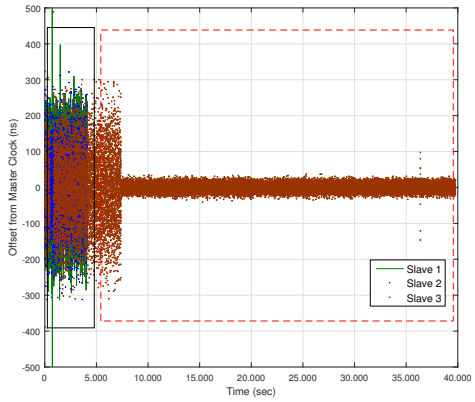

Fig. 9. Hardware Timestamping based Time Synchronization

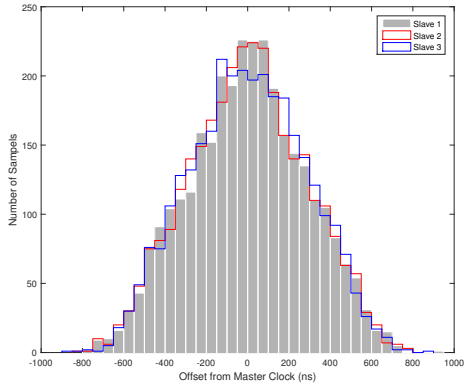

Fig. 12. Precision of Hardware-based PTP Implementation

50 to 50 ns, which is the highest accuracy achieved during all kind of test scenarios.

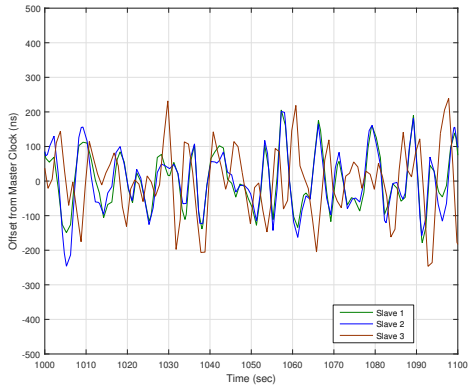

Fig. 10. Hardware Supported Time Synchronization (Multiple Slaves)

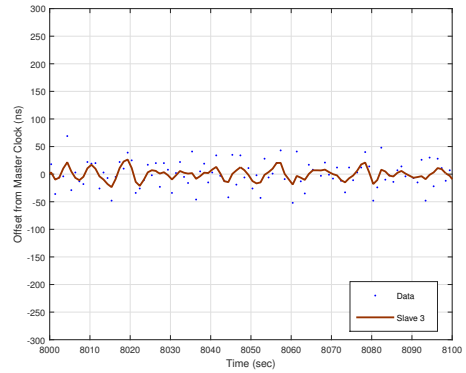

Fig. 11. Hardware Supported Time Synchronization (Single Slave)

The stability of clock can be noticed from figure 12, in most of the results the offset is between ± 400ns, with four PTP devices connected via LAN network. On the other hand, almost all slave clock have similar precision unlike software timestamping base PTP network, where all slave clocks have different behaviour in whole test case duration with similar configuration and environment.

*3) **Comparison of Software and Hardware-based Synchronization** :* The previous sections show that the hardware and software based synchronisation not only differ in offset from the master but also in precision. It can be seen in figure 13, after plotting software and hardware-based measurements on the same graph it is hard to distinguish that whether hardware timestamping is equal to zero or not.

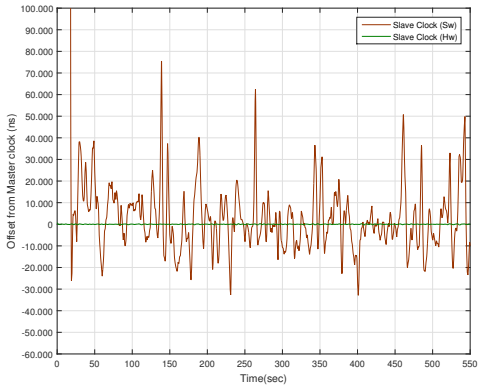

Fig. 13. Comparison of Software and Hardware based Synchronization

*4) **Hardware Assisted Time Synchronization under CPU Load**:* It was expected that the hardware-assisted time synchronisation would not have the prominent effect of CPU stress on the accuracy because the PTP messages are timestamped at the MAC/ PHY layer. Figure 14 compares the three slave with simulated CPU load by using the stress-ng tool. The slave 1 shows the normal synchronisation period, slave two devices are stressed using stress-ng with 50% utilisation parallel to the synchronisation process, and similarly, slave three is stressed with 100% CPU utilisation. In this test case, no considerable change in the behaviour of devices are

noticed, all the slave offsets are fluctuating between -300 to 300 nanoseconds.

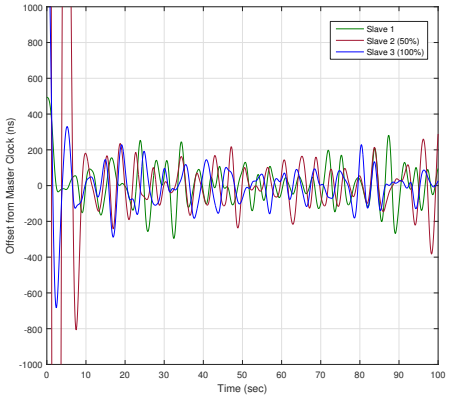

Fig. 14. Hardware Assisted Time Synchronization under CPU Load

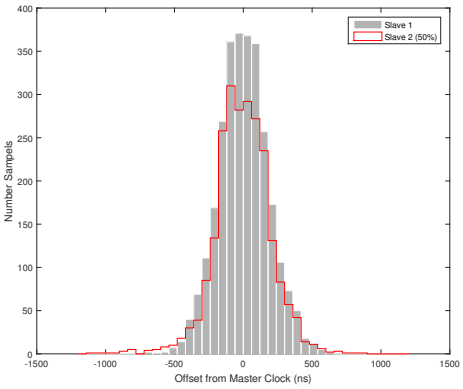

Fig. 15. Precision of Synchronization under CPU Load

We have tried to find out possible effects on the precision, but there is no considerable change found in the precision of the synchronisation. Figure 15 shows the comparison of two slaves, slave one is not experiencing any simulated load and slave two is stressed via 50% of CPU utilisation. There is little change visible in the precision of clocks, but this observation goes wrongs when we compare it with 100% load. So, it is concluded that there is no change observed in the precision of clock in current type of network hierarchy and environment. It is expected, there might be considerable effects in more stable PTP network hierarchy.

*5) **Hardware Assisted Time Synchronization under I/O Load**:* In order to perform I/O based test case scenario, I/O load type from stress-ng is used, which issues many tiny synchronous I/O reads and writes on a temporary file by utilising Linux *aio* interface. In this case, there is no considerable change found in accuracy and precision. Figure 16 shows the result, where both slaves (with and without load), are having the same accuracy.

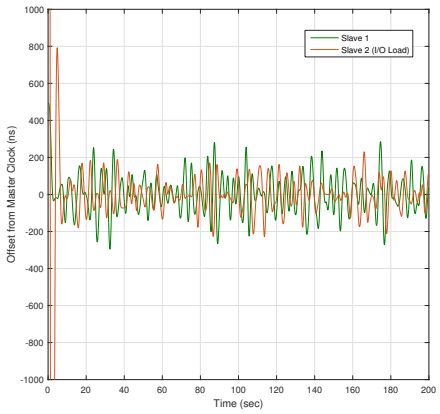

Fig. 16. Hardware Assisted Time Synchronization under I/O Load

*6) **Hardware Assisted Time Synchronization under Network Load**:* The effects of network-based load highly depend on the type of switches and routers are used in the network. In this study, the network components used to connect the slaves and master clock is not PTP supported. So, we need to consider that the results shown here can be different if more advanced networking devices will be used. In order to simulate the network load, an extra Linux based device is connected to the same network and *iPerf* tool is used to create a multicast stream with specified bandwidth. Additionally, in figure 17 and 20, the absolute value of data is considered in order to avoid the further entanglement.

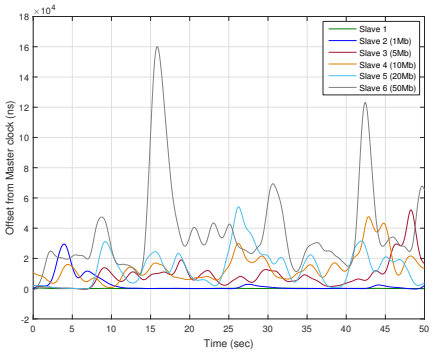

Fig. 17. Hardware Assisted Time Synchronization under Network Load

The figure 17 shows, slaves and their respective master clock expose to network traffic between the range of 0Mb to 50Mb. It can be seen that, as the amount of the network traffic grows the offset from the master clock also rises. Let assume that the 0-5Mb network load comes under the category of Low network load and 10-50Mb is the High load. First, we inspect the accuracy and precision in low network traffic and then High network traffic.

Figure 18 shows the offset calculations of slaves under low network load. During the synchronisation period of slave 1, there was no extra network traffic, and the accuracy was

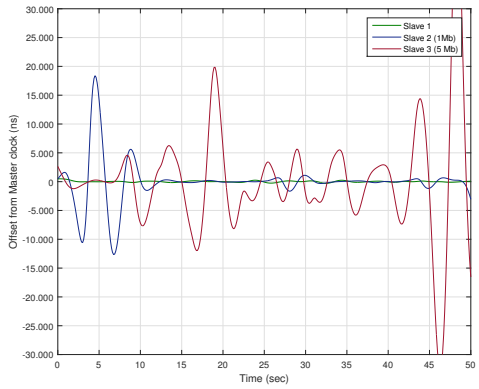

Fig. 18. Hardware Assisted Time Synchronization under Network Load (Low)

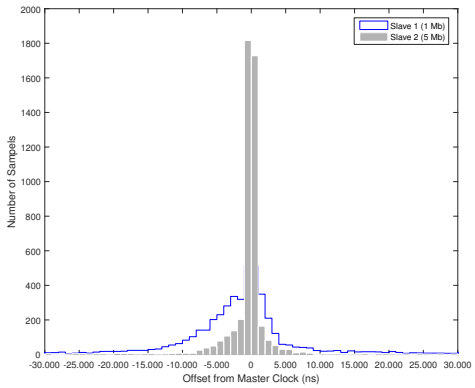

Fig. 19. Precision of Time Synchronization under Network Load (Low)

around ± 200 ns. On the other hand, while applying simulated network traffic of 1Mb and 5Mb on slave 2 and slave 3 the offset was remarkably advanced up-to ± 5us and ± 15us. The precision of both slave clocks is also dissimilar, as it can be seen in figure 19 where slave 1 exposed to 1Mb of network traffics, most to the time offset was between the normal range as compare to slave with network load but there are some measurements where offset went up to ± 5000ns. On the other hand, the slave with 5Mb network traffic is having very low precision and very uncertain behaviour, where the measurement samples are spread over the range of - 30000 to 30000ns.

In figure 20, three slaves are experiencing high network traffic, where the accuracy is reached similar to software timestamping based solution. The slave 4-5 which are having 10Mb and 20Mb of network traffic load, the offsets of these clocks reached up to ± 40000ns. On the other hand slave with 50Mb of network traffic have the worst accuracy. The figure 21 confirms that, as the bandwidth consumptions of UDP stream increases, the accuracy and precision of the clocks decreases.

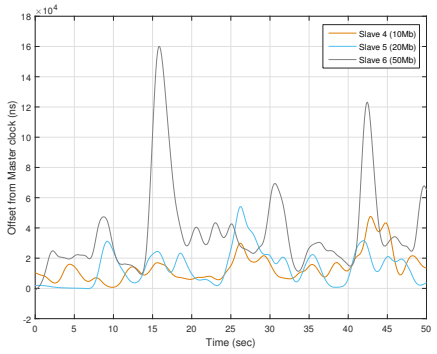

Fig. 20. Time Synchronization under Network Load (High)

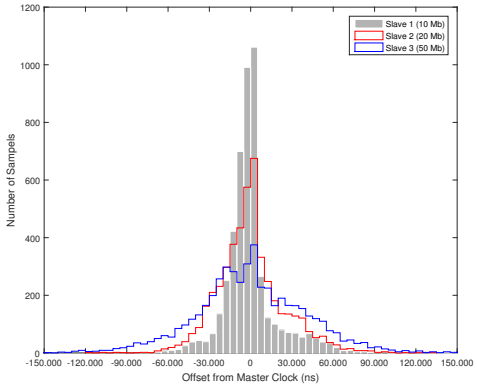

Fig. 21. Precision of Time Synchronization under Network Load (High)

VI. Conclusion

In software-based PTP synchronisation network, the average offset of slave clocks remained between ± 0.5ms, but there are some timespans where the offset of clocks reached to 2ms with multi-slave synchronisation model. Beside offset, the precision of slave clocks was also dissimilar with same configuration and communication medium.

In hardware timestamping based PTP implementations, the long-term results were very promising, where the accuracy of clock reached to approx 50ns. Overall in short-term measurements and random test cases, the accuracy of 200ns was frequently achieved. It is also observed that in a multi-slave synchronisation scenario, the slaves showed similar behaviour in term of accuracy and precision.

The tests with CPU and I/O based load do not show any noticeable change in accuracy of the clock on a hardware-based solution. There are also chances that the changes remained undetected due to smaller in scale. On the other hand, the test case scenarios with alien network traffic showed striking results, where the accuracy went to the lowest standard as compare other test case scenario including software timestamping based solutions.

Overall, the difference between the hardware and software based timestamping was significant. In load case scenarios,

apparently network traffic based tests showed some considerable effect. Expected effects from the CPU and I/O based tests either did not appear or we remained unable to detect them.

A. Improvements and Future Work

The tools used to simulate particular kind of load on PTP environment does not reflect any real-world patterns of the load, and it is possible with applying real-world resource exhausting application may show the different results. For example, there can be different types of network traffics that might affect the PTP setup in a different manner.

As discussed in earlier sections, the network component used in the PTP hierarchy establishment are unaware of PTP network, and there can be undetected effects on PTP network. Further study can be done in this area by comparing the results with PTP supported network component. The results of the simulated load based test case scenarios are only from hardware assisted PTP solution, there might be different results in a software-based solution.

REFERENCES

[1] IEEE Std 1588 -2002, IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems.

[2] IEEE Std 1588-2008 (Revision ofIEEE Std 1588-2002), IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems, July 24, 2008.

[3] R. Cochran and C. Marinescu, "Design and implementation of a ptp clock infrastructure for the linux kernel," in Precision Clock Synchronization for Measurement Control and Communication (ISPCS), 2010 International IEEE Symposium on, Sep. 27Oct. 1, 2010, pp. 116 121.

[4] R. Cochran, C. Marinescu and C. Riesch. "Synchronizing the Linux System Time to a PTP Hardware Clock," Precision Clock Synchronization for Measurement Control and Communication (ISPCS), 2011 International IEEE Symposium on 12-16 Sept. 2011.

[5] K. Correll, N. Barendt, and M. Branicky. "Design Considerations for Software Only Implementations of the IEEE 1588 Precision Time Protocol." in Proceedings of the IEEE 1588 Conference, Zurich, October 2005. vol. 1588, pp. 1115

[6] P. Ohly, D. N. Lombard, and K. B. Stanton, "Hardware assisted precision time protocol. Design and case study." in Proceedings of LCI International Conference on High-Performance Clustered Computing. Urbana, IL, USA: Linux Cluster Institute, 2008.

[7] A. Dreher and D. Mohl. "Precision Clock Synchronization IEEE 1588," White Paper by Hirschmann Automation and Control GmbH. Available: http://www.industrialnetworking.com/pdf/Hirschmann_IEEE_1588.pdf. [Accessed Aug 30, 2018]

[8] J.Ferrant, et al., "Synchronous Ethernet and IEEE 1588 in Telecoms: Next Generation Synchronization Networks." Wiley-ISTE, Jun 2013.

[9] D. W. Allan et al., "Precision oscillators: Dependence of frequency on temperature, humidity and pressure. " in Proceedings of the 1992 IEEE Frequency Control Symposium.

[10] J. Eidson. "A tutorial on IEEE-1588 Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems. " Tech. rep. Agilent Technologies, Inc, Oct. 2005. [Online]. Abailable: https://www.nist.gov/sites/default/files/documents/el/isd/ieee/tutorial-basic.pdf. [Accessed Aug 30, 2018]

[11] "AN-1838 IEEE 1588 Boundary Clock and Transparent Clock Implementation Using the DP83640." Tech. rep: SNLA104A by Texas Instruments Apr. 2013. Available: http://www.ti.com/lit/an/snla104a/snla104a.pdf.

[12] G. M. Garner, Overview and Timing Performance of IEEE Performance of IEEE 802.1AS," ISPCS 2008. Available: http://www.ieee802.org/1/files/public/docs2008/as-garner-mjt-ISPCS-2008-slides-0908.pdf.

[13] Steve T. Watt, S. Achanta, H. Abubakari, E. Sagen, Z. Korkmaz and H. Ahmed. "Understanding and Applying Precision Time Protocol." in Proceedings of the IEEE Conference on Power and Energy Automation Conference, Jeddah, 2014, pp. 23.

[14] Hyuntae Cho,Youngwoo Jin and Jusik Heo, "Implementation of a PTP Bridge to Extend IEEE 1588 to Zigbee Networks", Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on 29 June-1 July 2010

[15] Linux Kernel Documentation, Timestamping Control Interfaces. [Online]. Available: https://www.kernel.org/doc/Documentation/ networking/timestamping.txt. [Accessed Aug 30, 2018]

[16] Yocto Project (YP), "Yocto Project Documentation." [Online]. Available: https://https://www.yoctoproject.org/docs. [Accessed Aug 30, 2018]