


# Evaluation of Synchronization Protocols for fine-grain HPC sensor data time-stamping and collection

**Conference Paper****Author(s):**

Libri, Antonio; Bartolini, Andrea; Magno, Michele; [Benini, Luca](#) 

**Publication date:**

2016

**Permanent link:**

<https://doi.org/10.3929/ethz-b-000124772>

**Rights / license:**

[In Copyright - Non-Commercial Use Permitted](#)

**Originally published in:**

<https://doi.org/10.1109/HPCSim.2016.7568419>

# Evaluation of Synchronization Protocols for fine-grain HPC sensor data time-stamping and collection

Antonio Libri\*, Andrea Bartolini\*, Michele Magno\* and Luca Benini\*

\*Department of Information Technology and Electrical Engineering  
ETH, Zurich, Switzerland,  
{a.libri, barandre, michele.magno, lbenini}@iis.ee.ethz.ch

**Abstract**—Solutions for accurate and fine-grain monitoring are at the basis of the growth of future large-scale green high performance computing (HPC) infrastructures. The capability of these systems to adapt to specific application requirements relies on sensing and correlating several distributed physical parameters with application phases and states. Meeting such requirements allows thus to achieve a better use of the resources, higher throughput and higher energy-efficiency. As the capability of drawing such correlations relies on the synchronization across a network of nodes and measuring devices, the use of synchronization protocols becomes a critical component. Novel low-cost embedded devices start to include hardware support for network synchronization protocols to achieve a high resolution time accuracy. These devices are promising for monitoring physical parameters of HPC infrastructures. In this paper we evaluate how the performance of the two widely used network synchronization protocols, namely the Network Time Protocol and IEEE 1588, scale on a state-of-the-art embedded platform, namely a Beaglebone Black Board.

## I. INTRODUCTION

It is nowadays evident that the tradeoff between performance and energy consumption is a key challenge for future large-scale green HPC infrastructures [1]. Such infrastructures include a large variety of sensors for measuring architectural and physical run-time parameters. Both architectural and physical parameters have been historically used to understand applications performance and bottlenecks, as well as to monitor the status of the infrastructure by system administrators. With a view to increasing the total energy efficiency, there is an increasing demand for correlating applications and architectural events, measured from the processing elements, with physical parameters taken at the node level [1].

However, HPC applications usually run on multiple nodes [2], which are in the order of thousands or millions [3], each consisting of several processing elements and measuring points. As a result, the monitoring system can be seen as a multitude of agents that measure different metrics for the hardware (HW) components. Ultimately, the capability of correlating these monitoring points to form useful application metrics is bounded by their sampling rate as well as by the synchronization between the monitoring agents. Indeed, a synchronization in a distributed system is essential for the global ordering of events.

Distributed synchronization in IT systems is supported by network time synchronization protocols. In these protocols

all the nodes are kept synchronized against a time reference assumed as true time. In this context, accuracy refers to the amount of shift between the mean of the time-tags obtained by the agent and the time reference, while precision corresponds to the standard deviation. This is represented in Figure 1.

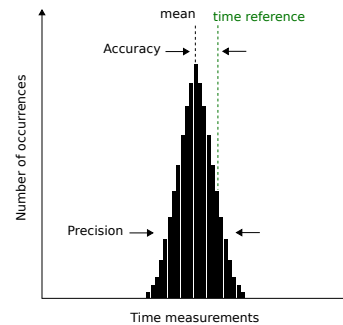


Fig. 1: Accuracy and precision of a set of measurements.

In the last few years two standards have emerged and are widely adopted: the Network Time Protocol (NTP) and the Precision Time Protocol (PTP), also known as IEEE 1588. The NTP was proposed by Mills in 1991 [4], and later revised in 2010 with the version 4 (NTPv4) [5]. It is targeted towards Wide Area Networks (WANs), where it typically achieves an accuracy in the order of a few milliseconds. However, as documented in the standard, within fast Local Area Networks (LANs) NTPv4 can reach a potential accuracy in the tens of microseconds. The Precision Time Protocol was proposed in 2002 and later revised in 2008 (PTPv2) [6]. It targets LANs and can synchronize devices with accuracy and precision in the sub-microsecond range. The protocol is suitable for measurement and control systems, and for applications where the cost of an external source of time for each node is not sustainable (e.g. using the Global Positioning System - GPS).

The Precision Time Protocol has a highly accurate and precise implementation, which is developed at CERN in collaboration with other partners. It is called White Rabbit (WR) [7], and it aims to be included into the next PTP standard revision. WR can synchronize nodes in an Ethernet-based network with sub-nanosecond accuracy and deterministic data transfer. Furthermore, it is based on an open-source paradigm for both its hardware and software implementation. However,

WR is not suitable when general purpose monitoring devices are used, as it requires specialized hardware. For this reason, we focus our analysis only on the NTP and PTP protocols.

A key concept in the study of such protocols is that synchronization is performed within the LAN of the HPC monitoring infrastructure. This allows to work in a corner-case for the NTP, where it can achieve its best performance. However, both of them need to be evaluated on the embedded low cost devices that are used for the HPC monitoring. With this goal, one of the best out-of-the-box solution to realize the fine-grain HPC monitoring, is a low-cost and low-power state-of-the-art embedded device, namely the Beaglebone Black Board (BBB). It is based on a TI Sitara AM335x processor, which is a 1GHz ARM Cortex-A8. The system-on-chip (SoC) has a built-in 12-Bit Successive Approximation Register (SAR) ADC, with 8-channels and a default sample rate of 200K Samples per second. Furthermore, it includes two Programmable Real-Time Units (PRUs), which make it suitable for on-board processing of the sampled HPC-sensor data. Finally, it is PTP hardware-enabled, which means it has a dedicated PTP hardware support that improves the synchronization accuracy and precision.

The main contribution of the paper is a fine performance evaluation of NTP and PTP in terms of accuracy, precision and scalability, when they are used in a context of large scale HPC monitoring. We conduct our analysis on a Beaglebone Black Board platform which is a promising device for smart monitoring and suitable for being integrated in a HPC infrastructure. Our study shows that: (i) NTP protocol achieves in our best configuration an accuracy of 17.5us and 8.4us of precision. This means that for 99% of the cases the timestamp offset (in between the monitoring device and its time reference) is below 35us.(ii) PTP protocol instead achieves in our best configuration an accuracy of 16.1ns and 513.7ns of precision. This means that for 99% of the cases the timestamp offset (in between the monitoring device and its time reference) is below 1.32us. (iii) when considering 75% of the cases the measured timestamp offset, in between the monitoring device and its time reference, becomes 23us for the NTP and 500ns for the PTP. (iv) time synchronization protocols do not represent the critical factor for the monitoring system scalability. Indeed, our results shows that, for the above mentioned values of time accuracy performance, these protocols require only 23 B/s per client for the NTP, and a traffic of 180B/s plus a data exchange of 186B/s per client for the PTP. In light of such results, our study demonstrates that both NTP and PTP, as well as the low-cost monitoring devices, can be used for fine-grain measurement of supercomputer systems.

The paper starts by introducing, in section II, the importance of HPC fine-grain monitoring and the key rule of synchronization in this context. Then, section III describes the selected synchronization protocols (i.e. NTP and PTP), in terms of their specifications (sections III.A and III.B) and implementations (sections III.C and III.D). Finally, section IV discusses the experimental results, focusing on the achieved accuracy and precision and further scalability. This is followed by a short "how-to" section, to outline the Beaglebone Black Board software settings, and conclusions.

## II. FINE GRAIN HPC MONITORING

This section introduces the concept of fine grain HPC performance monitoring and the importance of time synchronization in this context. Figure 2 sketches out a simplified picture of a HPC system.

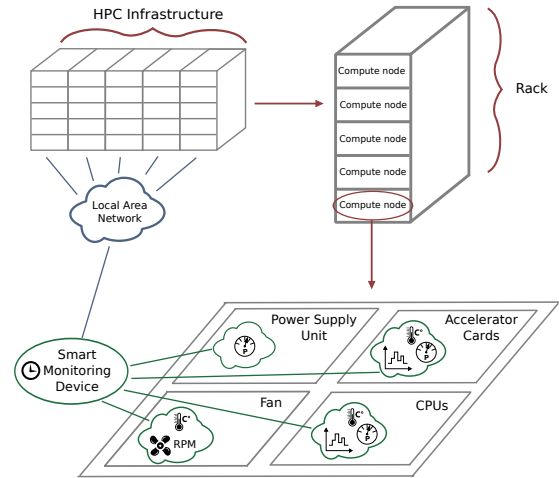


Fig. 2: Sketch of a HPC smart monitoring and data collection.

It consists of a set of racks composed by several nodes. HPC applications can run on one or multiple nodes at the same time, taking advantage of the various computational resources (e.g. several cores on different CPUs, accelerator cards, etc.). As effect of the computation, different physical parameters of the environment and of the components of the machine are modified. The analysis of these changes is important for system administrators and final users, and allows them to ensure safe working conditions, higher system performance and higher energy efficiency. A practical example is cooling, which is a costly operation on HPC machines. As a reference point, according to the TOP500<sup>1</sup>, nowadays most powerful supercomputer in the world, Tianhe-2, consumes ~17.8 MWs of peak power, that increases to ~24 MWs including the cooling infrastructure [8]. Works in [9][10] show that this cost can be reduced when advanced cooling control policies, based on extensive monitoring, are in place. Indeed, as depicted in Figure 2, several sensors are integrated in various node components. These include performance counters, temperature sensors, power gauges, etc. Therefore, a smart monitoring device, located on each node, it is at the basis of a distributed monitoring system. Such devices sample the data, process them in real-time (that explains the attribute "smart") and further sends them to a centralized unit via the LAN that connects the whole infrastructure. Correlating these measurements will lead to a better understanding awareness of the system, and will help modeling and later optimizing both the computational power and the energy efficiency of the entire machine.

This is where the synchronization protocols come into play. The key point is that the capability of correlating the data

<sup>1</sup>Top500 is an organization which ranks the 500 most powerful supercomputers in the world, using Linpack Benchmarks.

taken by different devices is bounded by both the sampling rate and the capability to precisely time stamp measurements. Indeed, every sampled data is associated with a timestamp. This is taken from a clock running on each embedded device. Therefore, the time offset between clocks is one of the main factor that bounds the monitoring granularity: the smaller the offset, the finer the monitoring and correlation grain.

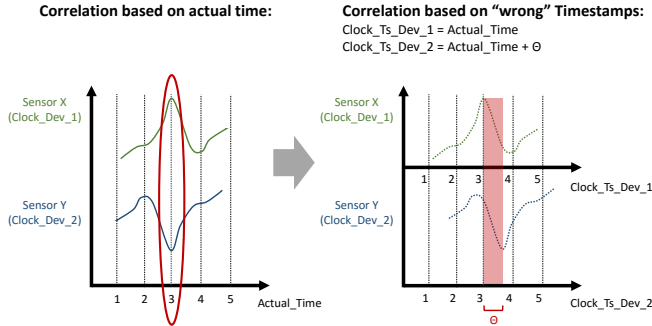


Fig. 3: Importance of the clock synchronization for a fine grain monitoring and events correlation.

Figure 3 explains the concept. Events occurred at the same time (represented with spikes in the left plot of the picture), could be interpreted in a wrong way if the timestamp offset  $\theta$  between the sampler devices (Clock\_Ts\_Dev\_1 and Clock\_Ts\_Dev\_2 in the plot on the right) is not "small" enough. We will see in the next sections that more than one clock can be present on a single device, and each of them introduces a different level of synchronization.

### III. SYNCHRONIZATION PROTOCOLS

In this section the key concepts of the two synchronization protocols used in this paper will be described. More in detail, the first two subsections are focused on their specifications, outlining the hierarchical topologies and the message synchronization patterns. In the last two parts their implementation is then introduced, focusing on the different timestamping methods and the main sources of jitter.

According to the protocols' standardization, from now on the terms client and server will be used in the NTP context, while slave and master in the PTP. However, there is no conceptual difference between them.

#### A. Hierarchical Topologies

Figure 4 shows typical hierarchical topologies for both NTP and PTP. Both protocols use a hierarchical master-slave network. In the NTP, each level is called "stratum" and can range from 0 to 15. Stratum 0 represents the time reference (e.g. atomic clock, GPS) and is connected to stratum 1. Lower strata are instead synchronized over the network, to the respective one-upper level stratum (e.g. stratum "n" to stratum "n-1"). The NTP algorithm sets then the synchronization paths by a shortest-path spanning tree with specific metrics. Devices in the same stratum can also peer with each other to stabilize the clock. Finally, stratum 16 indicates unsynchronized devices.

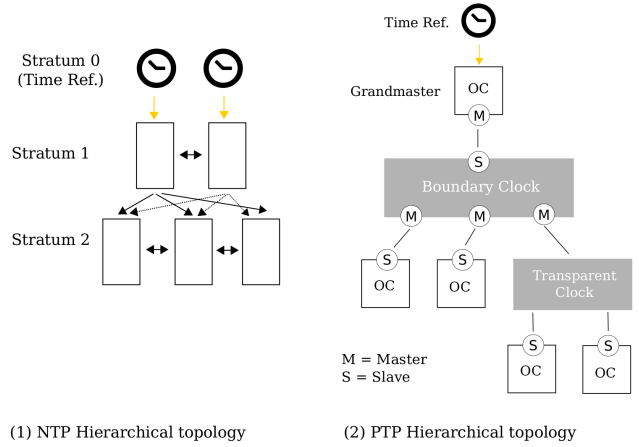


Fig. 4: Examples of hierarchical topologies.

The Precision Time Protocol uses a similar master-slave hierarchy within a LAN. Such a hierarchy is managed by the Best Master Clock (BMC) algorithm running on every device. As in a general network we can distinguish between end-nodes and networking-nodes (e.g. switches) used to interconnect the former. The end-nodes are devices with only one PTP port which can be either master or slave. Such devices are called *ordinary clocks* (OC). In particular, the ordinary clock that is the root timing reference for the whole PTP network is called *grandmaster clock*. This device is straight connected to a source of time which gives a high-degree of accuracy and precision (e.g. GPS). Therefore, to synchronize each network segment the BMC algorithm select one master between all the PTP ports. If the connection with the grandmaster is lost, other clocks may assume its rule.

PTP networking-nodes can be of two kinds. The first one is called *transparent clock* (TC), while the second one *boundary clock* (BC). Both have the goal to compensate the jitter (load dependent latency) introduced by general networking devices. The transparent clock corrects it by measuring the time taken for a PTP message to transit the device and adding that to the packet. It is "transparent" from the clocks point of view, as it does not have PTP ports which acts as a master or a slave for other nodes. Such a thing is done instead by the boundary clock, which has several PTP ports. As illustrated in Figure 4.2, in such a node one port (in a specific moment) is in a slave state, while the others are masters for one or more slaves.

#### B. Synchronization Message Exchange Pattern

The basic synchronization message exchange for both protocols is represented in Figure 5. The idea behind NTP is that each client regularly polls a cluster of servers to synchronize its clock, computing both the *round-trip delay*  $\delta$  and the *time offset*  $\theta$ . According to the standard, the latter is the time offset of the server relative to the client ( $\text{Time}_{\text{server}} - \text{Time}_{\text{client}}$ ). The message exchange pattern is composed by only a pair of messages: the `NTP_Req` from the client to the server (query) and the backward called `NTP_Resp` (reply). As soon as the server receives the query containing the client's transmission

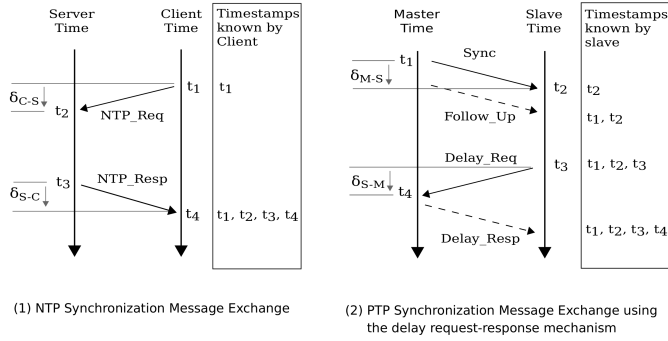


Fig. 5: Synchronization message exchange patterns.

timestamp  $t_1$ , it generates a reception timestamp  $t_2$  and sends it back together with its transmission timestamp  $t_3$  via the reply message. Therefore, the client notes its reply reception time  $t_4$ , and can finally compute both  $\delta$  and  $\theta$  by:

$$\delta = (t_4 - t_1) - (t_3 - t_2) \quad (1)$$

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

where equation 2 is calculated assuming:

- a symmetric network delay between the client (c) and the server (s),  $\delta_{c \rightarrow s} = \delta_{s \rightarrow c} = \frac{\delta}{2}$ ,
- by summing the server-client offset obtained by the NTP\_Req message  $\theta = t_2 - (t_1 + \frac{\delta}{2})$ , and that obtained by the NTP\_Resp message  $\theta = t_3 - (t_4 - \frac{\delta}{2})$ .

Equations 1 and 2 are also valid for PTP, where the one-way delay  $\frac{\delta}{2}$  it is used instead. Similar to NTP, the idea is to continuously exchange messages between master and slave ports in order to calculate both offset and one-way delay. Figure 5.2 describes the synchronization message exchange using the delay request-response pattern<sup>2</sup>. Four packets are involved: Sync, Follow\_Up, Delay\_Req and Delay\_Resp. Sync and Delay\_Req are called *event messages*, as an accurate timestamp is generated at both transmission and receipt. Instead, the other two are *general messages*, as no timestamp is required. That means only the event messages contribute to the actual computation of  $\theta$  and  $\frac{\delta}{2}$ , while the other two packets are used as support. Indeed, the master periodically sends Sync packets to the slaves, taking note of its transmission current time  $t_1$ . Such a time is then delivered within the general message Follow\_Up. The reason behind this mechanism is that, to include  $t_1$  within the Sync packet itself, dedicated hardware is necessary. In the PTP nomenclature, such kind of hardware is called one-step clock, while the generic hardware which delivers the time in two steps (Sync + Follow\_Up) is called two-step clock. As soon as the slave receives the Sync message, it notes the reception time  $t_2$ , and conveys the Delay\_Req packet to the master, writing down its transmission time  $t_3$ . In the last step, the slave receives from the master

<sup>2</sup>A second option would be to evaluate the protocol synchronization performance via the peer delay mechanism pattern, which is actually less flexible, and for such a reason left for possible future works.

the Delay\_Req reception time  $t_4$  by the Delay\_Resp message. Therefore, the slave can finally measure both offset and mean propagation time.

Some further considerations on the PTP synchronization pattern are essential to understand its network scalability, which will be later examined. The key concept is that PTP communication is based on a multicast messaging model (or potentially a unicast messaging model). In other words, each PTP packet sent by any PTP port has a destination's multicast address, which means that will be received by all the PTP ports in the network segment. For packets which are specific for a device (e.g. the Delay\_Resp message, which is specific for the slave that delivered the Delay\_Req), then the clock identity of the destination device is specified within the message. This allows PTP to reduce the network load on the master side.

### C. Timestamping Methods

One of the main sources of jitter that directly affects the time synchronization accuracy is due to the uncertain processing time of the protocol stack during the packets transmission. Ideally, the packet is forwarded right after the timestamp is generated, with no related timing delay. However, in the real implementation, this delay depends on the layer of the protocol stack where the timestamp is done: the higher the distance from the physical layer, which is the exact point where the packet is transmitted, the greater the jitter introduced in the timestamp. Figure 6 depicts the difference between ideal and real implementation, and the several points where the timestamp can be done.

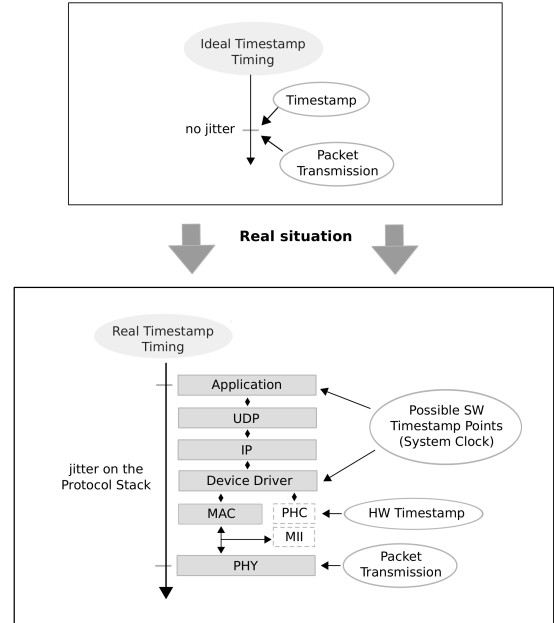


Fig. 6: Sources of jitter in the protocol stack.

The two possibilities are the *software timestamp* and the *hardware timestamp*. The latter is available only for PTP hardware-enabled devices, and it is actually one of the main advantages that PTP has over NTP. The software timestamp, indeed, is the least accurate option. It is based on a software

clock that runs in the kernel, namely the *system clock*. It can be applied when a packet reaches the application layer or, in order to reduce the delay in the protocol stack, the device driver layer. The system clock works with timer interrupts and keeps the time by reading the CPU register which counts the number of clock cycles since the last reboot (e.g. the Time Stamp Counter on Intel x86 processors). The system clock, has not to be confused with the battery powered clock, also known as Real-Time Clock and present in most Linux devices. This clock, indeed, is used only to keep track of the time when the system is turned off and later initialize the system clock at boot time. In Linux systems, NTP is usually implemented by a daemon running in user space, the *ntpd*, which constantly updates the system clock. The kernel will then correct the real-time clock drift, usually at a much lower frequency.

To minimize the jitter introduced by the processing time of the OSI<sup>3</sup> layers, PTP introduces the idea of hardware timestamping support. As described in Figure 6, the *PTP Hardware Clock* (PHC) subsystem, sketched here with the PHC block, takes advantage of the Media Independent Interface (MII) to detect PTP frames and provide timestamps with an accuracy close to the physical layer. Several Linux implementations are available for PTP. The one used in this paper is the Linux PTP Project [11]. It involves two user space applications, *ptp4l* and *phc2sys*. Both take advantage of the kernel space support for the PHC subsystem by using the *clock\_gettime* family of calls. While *ptp4l* is the actual implementation of the PTPv2, implementing both boundary and ordinary clocks, *phc2sys* is used to synchronize the PHC to the system clock.

We will see in the next sections, it is possible to tune the synchronization messages rate on both protocols implementations, *ntpd* and *ptp4l*. This can be done by setting the client (slave in the case of PTP) polling period parameter. We will see also that while it is useful to increase such frequency for the NTP, it is not for the PTP. Instead, using *phc2sys* to increase the update rate between the PHC and the system clock, will result in a higher synchronization performance.

#### D. Possible Sources of Jitter

Aim of this subsection is to summarize the main sources of jitter that affect the time synchronization performance that will be after evaluated. The main difference between the two protocols is of course the method used for the timestamp, as the HW timestamping support reduces the processing time delay to the OSI physical layer. Therefore, remaining sources of jitter, on both protocols, are mainly attributed to [12][13]:

- 1) the delay on the physical link, which includes
  - the physical channel (i.e. asymmetry of the network propagation delay in the two directions),
  - the use of general networking devices. In the PTP that can be improved by using PTP-enabled networking devices. In the NTP by replacing general switches with more recent models, which can be used as NTP time server. Indeed, the number of

hops between client and server will be reduced to a point-to-point connection.

- the physical distance between the PHY layer and the PHC support within the device, for PTP only.
- 2) the hardware properties of the clock (i.e. rate and stability of the oscillator, which result in a limited hardware resolution and precision of the timestamps, respectively). More in detail,
    - in the NTP, the rate and the stability of the oscillator used for the system clock interrupt timer.
    - in the PTP, the rate and the stability of the PHC.

## IV. EXPERIMENTAL RESULT

### A. Testbed Setup

The goal of this section is to evaluate how the performance of NTP and PTP scale on a Beaglebone Black Board used as monitoring device in this work. For this purpose, we set up the testbed described in Figure 7. It consist of a slave node (BB1) directly connected to a master (BB2). Using the NTP nomenclature, BB1 is the client and BB2 the server. We used it to find an upper bound (best performance) for the achievable accuracy and precision, aimed at the sensor data timestamping and collection within a HPC infrastructure. Moreover, in this scenario it is not necessary to synchronize clocks with an absolute time reference, but instead, it is important that all the collected sensors data are synchronized between each other and the rest of the HPC nodes. Therefore, the system clock of the Beaglebone Black master is used as source of time:

- In the NTP tests, the client system clock is directly synchronized to server system clock by the daemon *ntpd*.
- In the PTP tests, the *ptp4l* application, running on both Beaglebones, synchronizes the two hw ordinary clocks (PHC slave to the PHC master), while the *phc2sys* program updates the two system clocks with the respective PHC. In particular, the *phc2sys* running on BB1, synchronizes its system clock taking the time from its PHC. Instead, the *phc2sys* on BB2 does the opposite, updating its PHC taking the time from its system clock.

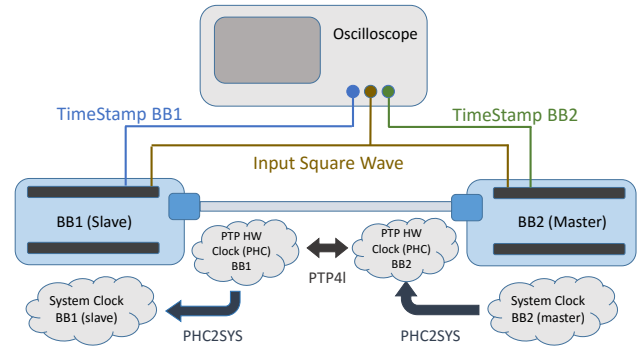


Fig. 7: Testbed setup which sketches PHC and system clock running on both Beaglebones, master and slave. In the case of NTP, only the system clock is used.

<sup>3</sup>ISO/OSI model, which stands for International Organization for Standardization/Open Systems Interconnection model.

To have an empirical measurement of the time precision and accuracy in-between the two embedded devices, we had to solve two technical problems: (i) the two Beaglebones need the same triggering event. (ii) We have to measure with an external reference the actual time when the two devices sense the triggering event and generate the timestamp.

The idea is to use an input square wave as source of interrupt for triggering both nodes. The GPIO used as input pin, is then handled through the Interrupt Service Routine (ISR) within an ad hoc device driver. As soon as the square wave goes high, a second GPIO, used as output pin and connected to an oscilloscope (Agilent MSOX3054A), is driven high. Hence, in the next instruction a timestamp is generated, taking the time from the system clock. Figure 8 outlines the oscilloscope window.

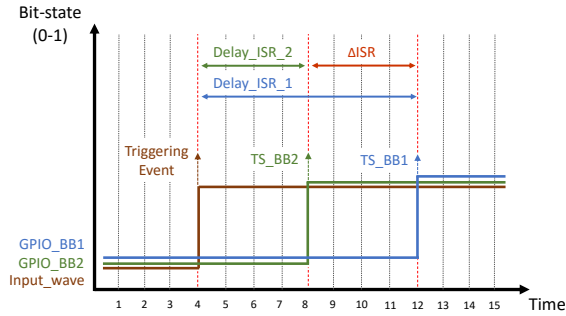


Fig. 8: Oscilloscope’s window that shows the several involved signals (i.e. input wave and the two output GPIOs on both Beaglebones), and the delays introduced by the interrupt service routines in-between the triggering event and the actual moment the timestamp is generated (i.e. Delay\_ISR\_1 and Delay\_ISR\_2).

The blue line represents the output GPIO on the Beaglebone slave (GPIO\_BB1), while the green line the one on the master (GPIO\_BB2). As soon as the two Beaglebones receive the trigger event, they will generate a timestamps (TS\_BB1 and TS\_BB2), both with a ISR processing time delay (Delay\_ISR\_1 and Delay\_ISR\_2). Therefore, such configuration allows to take into account the offset between the two ISR processing time delays ( $\Delta_{ISR}$ ), for the final computation of the system clock time-offset:

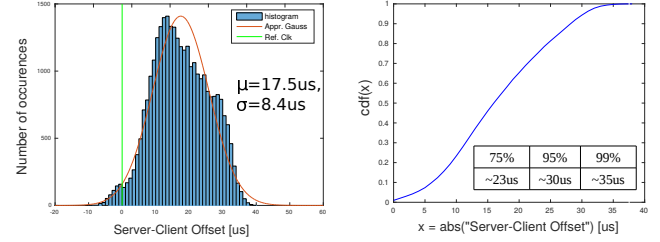
$$System\_Clock_{offset} = TS_s - (TS_m + \Delta_{ISR}) \quad (3)$$

where  $TS_s$  and  $TS_m$  correspond to system clock timestamps of the Beaglebone slave and master, respectively. Moreover, the offset  $\Delta_{ISR}$  is here always referred to the master time and can be either positive or negative depending on which clock is head of time.

## B. Measurement Results

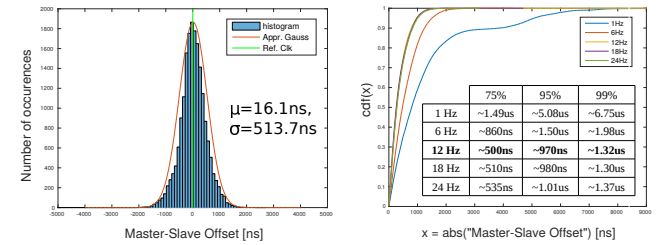
1) *Accuracy and Precision:* Both values were traced by measuring the skew between the two system clocks, for several working frequencies of the previously mentioned Linux

programs (i.e. ntpd, phc2sys, and lastly ptp4l). Regarding NTP, the ntpd polling period can be set by the two options minpoll and maxpoll within the range 8s to ~36.4h. Default values are 64s for minpoll and 1024s for maxpoll. The histogram in Figure 9.1 shows the master-slave offset obtained by setting both values to the minimum polling period of 8s.



(1) NTP: Histogram of the server-client offset.

(2) NTP: Cumulative Distribution Function



(3) PTP: Histogram of the master-slave offset.

(4) PTP: Cumulative Distribution Function

Fig. 9: Best tested NTP and PTP synchronization performance on a Beaglebone Black Board.

The green line on the zero corresponds to the master clock, while the red curve is the approximated Normal Distribution. As can be seen in the plot, the measured accuracy (mean value) over 30k samples is ~17.5us, and the precision (standard deviation) is ~8.4us. To observe how the percentage of samples skews from the reference time, the Cumulative Distribution Function (CDF) is traced in Figure 9.2. Results show that 75% of the samples stay within ~23us, 95% below ~30us, and finally 99% of the values below ~35us. Note that this exchanging message rate should not be a problem within a HPC LAN. Moreover, the scalability factor will be later analysed.

Figure 9.3 shows the best PTP performance trade-off. The phc2sys program was used to tune the internal-slave clock update rate on both Beaglebones, namely the PHC update rate on BB2 and the system clock update rate on BB1. This is not related to the frequency of synchronization messages exchanged over the network, which instead is possible to set with the ptp4l application. The reason for not altering such a frequency is that its default configuration in ptp4l is already set to the maximum value of 1Hz for all the event messages. Moreover, there would not be any reason to further increase this frequency as it is the typical best trade-off to achieve the minimum traffic on the network and the optimal point of work of the PHC oscillator [12]. The tested frequencies range from 1Hz to 24Hz with steps of 6Hz. Table I reports such results, while the CDFs in Figure 9.4 stand out that 12Hz

corresponds to the best trade-off to minimize both frequency and skew from the time reference. Indeed, results show that 75% of the values are below  $\sim 500\text{ns}$ , 95% below  $\sim 970\text{ns}$ , and finally 99% below  $\sim 1.32\mu\text{s}$ .

Update-rate	$\mu$ -offset	$\sigma$ -offset
1Hz	-230.8ns	2.01us
6Hz	-9.2ns	753ns
<b>12Hz</b>	<b>16.1ns</b>	<b>513.7ns</b>
18Hz	101.6ns	458.7ns
24Hz	23.6ns	488.5ns

TABLE I: PTP performance achieved by tuning the phc2sys internal-slave clock update rate to several frequencies. Accuracy and precision are indicated here by  $\mu$ -offset and  $\sigma$ -offset, respectively.

As both protocol implementations are based on free-running oscillators [14], decreasing the synchronization message frequency over the network it follows in a growth of the time drift between the master and the slave Beaglebones. Of course, such a drift is bounded within two consecutive synchronizations and related to the system clocks in the NTP case, and PHCs in the PTP (which actually follows in a drift of the respective master and slave system clocks). The four plots in Figure 10, obtained by setting the maximum polling period (i.e.  $\sim 36.4\text{h}$ ) on both ntpd and ptp4l, give an idea of such a drift.

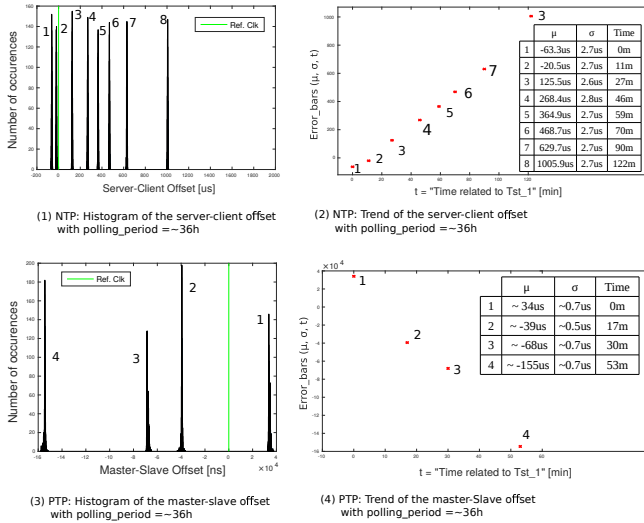


Fig. 10: Linear drift between the two Beaglebone free-running oscillators.

In particular, Figure 10.1 and Figure 10.3 reports the several histograms for NTP and PTP, respectively. The error bars in Figure 10.2 and Figure 10.3 show instead the almost linear trend of both drifts. Indeed, the x-axis represents the elapsed time between the first test and the others, and the fact that the trend is growing in a positive or negative direction depends on which clock was ahead of time during the initial synchronization.

In light of the achieved results, the Network Time Protocol running on a Beaglebone Black Board allows a time-synchronization for a fine grain HPC monitoring, and further

events correlation, with an accuracy of  $\sim 17\mu\text{s}$  and precision of  $\sim 8.4\mu\text{s}$ . Such values decrease to  $\sim 16\text{ns}$  and a  $\sim 513\text{ns}$ , respectively, using the Precision Time Protocol. Furthermore, the results track an upper bound of  $\sim 35\mu\text{s}$  and  $\sim 1.32\mu\text{s}$  on NTP and PTP, respectively, for 99% of cases.

2) *Scalability*: Looking at the message exchange patterns, it is possible to quantify, for both protocols, how the master scales with the number of connected devices. In particular, considering  $N$  clients, the NTP server (bottleneck) has to deal with  $2N$  packets per time update (a pair query-reply). In the NTPv4, the query consists of 90Bytes (42B for the header and 48B for the payload), while the reply involves 94Bytes (46B the header + 48B the payload) [5]. Therefore, with a polling period of 8s (our best tested performance) the NTP server data rate corresponds to 23B/s per client, where of course the best case in terms of such a data rate corresponds to the client-server point-to-point link.

Focusing on the PTP multicast messages exchange model, the master has to handle  $2 + 2N$  packets per time update. In other words, setting a polling period of 1s for all the synchronization messages, there is a fixed component of 180B/s (90B/s for the Sync + 90B/s for the Follow\_Up), and a scalable component of 186B/s per slave (86B/s for the Delay\_Req + 100B/s for the Delay\_Resp) [6]. To be precise, PTP provides other general messages with the view to handle the PTP network. These messages are sent with a lower frequency and mainly contribute to the fixed component only, reason why they do not undermine the scalability. Such considerations are valid for network topologies with transparent clocks (or generic non-PTP devices) between the two nodes master and slave. Again the best case in terms of such a data rate corresponds to the master-slave point-to-point link, which could be achieved, for instance, using boundary clocks between the two nodes.

With the goal to synchronizing devices within a monitoring infrastructure, we can finally assert that both protocols are not critical in terms of scalability. Indeed, considering a Fast Ethernet, which support the bit rate of 100Mb/s (e.g. the 10/100 RJ45 of the Beaglebone Black Board), in our best performance configuration the NTP server uses only  $\sim 0.000184\%$  of the network bandwidth per client, while the PTP master only  $\sim 0.001488\%$  of the network bandwidth per slave (where in case of a network with only boundary clocks between the several nodes there is only one slave per master). In theory, using only 10% of the bandwidth a NTP server could handle up to  $\sim 54\text{k}$  nodes, while each PTP master port up to  $\sim 6.7\text{k}$  nodes. Moreover, using a Gigabit Ethernet these values decrease to  $\sim 0.000184\%$  and  $\sim 0.0001488\%$  for the NTP server and PTP master, respectively. As before, in theory this would correspond to handling up to  $\sim 540\text{k}$  nodes for each NTP server and  $\sim 67\text{k}$  nodes for each PTP master port.

### C. Beaglebone Black Board - NTP-PTP Settings

In this final section we describe a short "how-to", based on the experience gained in this work, in order to achieve our best time synchronization results on the BBB. Table II summarizes the main settings of both ntpd and phc2sys, for NTP and PTP, respectively.



Program	Server (Master)	Client (Slave)
ntpd	minpoll 3 maxpoll 3 fudge stratum n	minpoll 3 maxpoll 3 fudge clock_IP stratum n
phc2sys	-R12	-R12

TABLE II: Ntpd and Phc2sys settings to achieve our best accuracy and precision on a Beaglebone Black Board.

In the NTP configuration file (`/etc/ntp.conf`) it is possible to set the polling period of both client to its server, and server to its time-reference. Furthermore, the option `fudge` allows to specify the stratum number (from 0 to 15) associated to the source of time in both client and server. Regarding PTP, the `-R` parameter allows `phc2sys` to set the polling rate of the internal-slave clock (PHC or System Clock). Moreover, it is possible to generate the System Clock timestamp using the `clock_gettime` family of calls. In particular, within the device driver `do_gettimeofday()` was used. Otherwise, in a user-space program the `gettimeofday()` call can be used. Note that, using instead `clock_gettime()` along with the `CLOCK_REALTIME` tag, results in a performance deterioration. Indeed, such timestamp is generated by the Real-Time Clock previously described, which is updated with a lower frequency than the system clock. Finally, if synchronization with an absolute-time reference is needed, we suggest the use of a GPS at the top of the hierarchical topology, instead of using a cluster of servers outside the LAN. Indeed, this could decrease the performance due to the jitter introduced by typical store-and-forward networking devices. We remark that our analysis is based on synchronization with a relative-time reference as we are interested in application and sensor data correlation of devices within a HPC infrastructure LAN, and not outside of it.

## V. CONCLUSION

Solutions for accurate and fine-grain monitoring are crucial for the growth of large-scale green HPC infrastructures. These large scale systems require a distributed monitoring and thus the time-granularity, at which architectural and physical events can be correlated and analysed, is bounded by the monitoring devices' time-synchronization. In this work we evaluate the achievable synchronization performance in terms of accuracy, precision and scalability of two widely adopted time synchronization protocols: the Network Time Protocol and the Precision Time Protocol, both running on a state-of-the-art embedded monitoring platform, namely the Beaglebone Black Board. Our results show that the NTP achieves, in our best configuration, an accuracy of  $\sim 17.5\mu\text{s}$  and a precision of  $\sim 8.4\mu\text{s}$ , while the PTP decrease such values to  $\sim 16.1\text{ns}$  and  $\sim 513.7\text{ns}$  respectively. In 99% of the performed tests, the timestamp offset calculated between the monitoring device and its time reference is not greater than  $\sim 35\mu\text{s}$  for the NTP and not greater than  $\sim 1.32\mu\text{s}$  for the PTP. Furthermore, in 75% of the cases, such a timestamp offset decreases to  $\sim 23\mu\text{s}$  for the NTP and  $\sim 500\text{ns}$  for the PTP. Finally, given these values of time synchronization, these protocols are not critical to the monitoring system scalability.

As a result, our study demonstrates that both NTP and PTP, as well as low-cost embedded monitoring devices, can be used for fine-grain measurement of HPC systems. Furthermore, such results can be applied in any context of performance monitoring within a LAN that takes advantage of a Beaglebone Black (more in general that uses the TI Sitara AM335x processors' Family) and PTP-HW enabled devices for PTP. We also expect that using different processors as monitoring devices, as long as they are PTP-HW enabled, will not change the order of magnitude of the results. Finally, as the state-of-the-art of some built-in HPC performance monitoring sensors (e.g. power sensors) do not have a sub-milliseconds resolution, our study shows that synchronization is not a boundary factor for application and sensor data correlation. Moreover, in our future work we plan to investigate faster power monitoring strategies and their implication in large scale systems.

## ACKNOWLEDGMENT

This work was partially supported by the EU H2020 FET-HPC project ANTAREX (g.a. 671623) and by the FP7 ERC Advance project MULTITHERMAN (g.a. 291125).

## REFERENCES

- [1] T. Ilsche, D. Hackenberg, S. Graul, R. Schöne, and J. Schuchart, "Power measurements for compute nodes: Improving sampling rates, granularity and accuracy," in *Green Computing Conference and Sustainable Computing Conference (IGSC), 2015 Sixth International*, Dec 2015, pp. 1–8.
- [2] A. Bartolini, M. Cacciari, C. Cavazzoni, G. Tecchiolli, and L. Benini, "Unveiling Eureka - 2014; thermal and power characterization of the most energy-efficient supercomputer in the world," in *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*, March 2014, pp. 1–6.
- [3] ETP4HPC, "Strategic Research Agenda 2015 (Update)," <http://www.etp4hpc.eu/image/fotos/2016/01/ETP4HPC-SRA-2-Single-Page.pdf>, 2015.
- [4] D. L. Mills, "Internet time synchronization: the network time protocol," *IEEE Transactions on Communications*, vol. 39, no. 10, pp. 1482–1493, Oct 1991.
- [5] J. Burbank, W. Kasch, and P. D. L. Mills, "Network Time Protocol Version 4: Protocol and Algorithms Specification," IETF RFC 5905, Oct. 2015. [Online]. Available: <https://rfc-editor.org/rfc/rfc5905.txt>
- [6] "IEEE standard for a precision clock synchronization protocol for networked measurement and control systems," *IEEE Std 1588-2008 (Revision of IEEE Std 1588-2002)*, pp. 1–269, July 2008.
- [7] <http://www.ohwr.org/projects/wr-std/wiki/>, [Online: 30-March-2016].
- [8] J. Dongarra, "Visit to the National University for Defense Technology Changsha, China," Technical report, University of Tennessee, June 2013.
- [9] C. Conficoni, A. Bartolini, A. Tili, G. Tecchiolli, and L. Benini, "Energy-aware cooling for hot-water cooled supercomputers," in *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2015, pp. 1353–1358.
- [10] A. Borghesi, C. Conficoni, M. Lombardi, and A. Bartolini, "Ms3: A mediterranean-stile job scheduler for supercomputers - do less when it's too hot!" in *High Performance Computing Simulation (HPCS), 2015 International Conference on*, July 2015, pp. 88–95.
- [11] <http://linuxptp.sourceforge.net/>, [Online: 30-March-2016].
- [12] P. Loschmidt, R. Exel, A. Nagy, and G. Gaderer, "Limits of synchronization accuracy using hardware support in ieee 1588," in *Precision Clock Synchronization for Measurement, Control and Communication, 2008. ISPCS 2008. IEEE International Symposium on*, Sept 2008, pp. 12–16.
- [13] M. Lipiński, T. Włostowski, J. Serrano, and P. Alvarez, "White rabbit: a ptp application for robust sub-nanosecond synchronization," in *Precision Clock Synchronization for Measurement Control and Communication (ISPCS), 2011 International IEEE Symposium on*, Sept 2011, pp. 25–30.
- [14] J. Serrano, M. Cattin, E. Gousiou, E. van der Bij, T. Włostowski, G. Daniluk, and M. Lipiński, "THE WHITE RABBIT PROJECT," in *Proceedings of IBIC2013, Oxford, UK, 2013*.