Time Synchronization Services for Low-Cost Fog Computing Applications

Peter Volgyesi, Abhishek Dubey, Timothy Krentz, Istvan Madari, Mary Metelko, Gabor Karsai
peter.volgyesi@vanderbilt.edu
Institute for Software Integrated Systems, Vanderbilt University
1025 16th Ave South
Nashville, TN 37212, USA

ABSTRACT

This paper presents the time synchronization infrastructure for a low-cost run-time platform and application framework specifically targeting Smart Grid applications. Such distributed applications require the execution of reliable and accurate time-coordinated actions and observations both within islands of deployments and across geographically distant nodes. The time synchronization infrastructure is built on well-established technologies: GPS, NTP, PTP, PPS and Linux with real-time extensions, running on low-cost BeagleBone Black hardware nodes. We describe the architecture, implementation, instrumentation approach, performance results and present an example from the application domain. Also, we discuss an important finding on the effect of the Linux `RT_PREEMPT` real-time patch on the accuracy of the PPS subsystem and its use for GPS-based time references.

CCS CONCEPTS

•Hardware →Smart grid; •Computer systems organization *→Real-time system architecture;*

KEYWORDS

Time Synchronization, Smart Grid, Fog Computing, GPS, PTP, Real-time Systems

ACM Reference format:
Peter Volgyesi, Abhishek Dubey, Timothy Krentz, Istvan Madari, Mary Metelko, Gabor Karsai. 2017. Time Synchronization Services for Low-Cost Fog Computing Applications. In *Proceedings of RSP'17, Seoul, Republic of Korea, October 15–20, 2017,* 7 pages.
DOI: 10.1145/3130265.3130325

1 INTRODUCTION

The Smart Grid is envisioned as a complex, distributed embedded system that brings computation and communication into power system engineering. In a Smart Grid, for example, smart meters installed at homes will autonomously buy and sell energy from and to other prosumers who are themselves producers and consumers of energy, while overall monitoring and control systems autonomously maintain grid stability. Obviously, such functions necessitate sophisticated embedded software and hardware systems that are resilient to faults and cyber attacks.

Arguably, the embedded software for such applications needs to be built on a robust software platform that provides programming abstractions and specific services for distributed and decentralized applications that run on a network of computing nodes. Such a platform can be called a fog computing platform, in the sense that it serves as a basis for running applications closer to the 'ground' instead of in the 'cloud'. This allows the computing power to be closer to the sensors and actuators where time constraints impact capability. Our team is working on such a platform, called Resilient Information Architecture Platform for Smart Grid (RIAPS) [13].

One of the core capabilities of such a platform is a highly accurate time synchronization service that ensures that the system clocks of the individual nodes are synchronized. This is necessary as it is possible that not all nodes have an attached GPS receiver (that would otherwise provide a GPS time base), and the nodes have to rely on a network-clock for their work. Furthermore, particularly in power systems, sensor data often have to be time-stamped in order to get an accurate picture of the state of the system. Time-critical systems are also sensitive to the time spent in calculations, so knowing what the time is when a calculation is finished could be important.

When prototyping such time-sensitive applications one can use a common master clock (say, a shared GPS receiver that emits a 1 PPS signal), but this is different from the fielded system where not all nodes have GPS. Hence, even for prototyping and development one needs a network-based time synchronization mechanism so that the testing happens in a realistic environment. In this paper, we describe such a time synchronization service, its hardware support and its software implementation that could be used both for prototyping and deployment in real systems.

The organization of this paper is as follows. First we discuss the technological building blocks for the service, next we introduce the fundamental concepts used in the design. This is followed by a description of the implementation and its evaluation. Next, a practical application example is described that shows how the service can be used in decentralized applications where high-precision time is required. The paper concludes with a summary and plan for future work.

2 BACKGROUND

The targeted application domain presents critical requirements for minimizing the jitter and uncertainty when executing timed, node-local actions. We evaluated several run-time options—using

RSP'17, Seoul, Republic of Korea

DOI: 10.1145/3130265.3130325

the `cyclictest` [1, 3] tool for task scheduling measurements and custom code for testing the timing accuracy of I/O actions. In almost all cases, the `RT_PREEMPT` [10] Linux real-time extensions—an extensive patch set for making the Linux kernel fully preemptible—resulted in superior performance.

Satisfying time-deterministic requirements on a single node is essential, but not enough for distributed systems or even for single nodes that must interact with the physical environment on a global timescale. In those cases we need to establish a common synchronized time base and need to align each node's local clock(s) to this global reference. Even slight differences in each node's local clock—typically a few tens of parts per million (ppm)—accumulate fast and become apparent over time. Based on environmental factors (temperature, humidity, and voltage stability), the frequency differences are not constant. Thus, to provide an accurate globally synchronized time base, the supporting services need to periodically measure and compensate for these differences. This periodic adjustment of the local time on the node requires careful considerations to avoid disruption of the local event scheduler. Fortunately, there are two, well-established technologies for solving this problem.

The Network Time Protocol (NTP) is a ubiquitous time synchronization service using heuristic software algorithms with no special requirements on the networking hardware and communication infrastructure. The Precision Time Protocol (PTP, IEEE-1588), on the other hand, is built on accurate end-to-end hardware-level timestamping capabilities. It is no surprise that the attainable accuracy of the two methods differ by orders of magnitudes: tens of milliseconds with NTP vs. microseconds with PTP [8]. Considering the timing requirements of the targeted application scenarios and the expected performance of node-level time determinism, we selected PTP as the primary approach for time synchronization among the nodes.

PTP achieves excellent accuracy (on the microsecond scale) if used within a local area network and/or all network equipment in the packet forwarding path participate in the protocol. The basic building blocks of the protocol are: (1) a hierarchical master/slave clock tree strategy supported by a leader-election (*best master*) protocol, (2) accurate time-of-flight measurement of network packets with the built-in assumption that these delays are symmetrical (3) support for measuring and compensating for intermediate delays across the communication medium (4) using link-level LAN frames or IPv4/IPv6 UDP messages as the transport mechanism (5) support for co-existing independent PTP clock domains on the same LAN. In Linux, it is implemented by the Linux PTP [5] project with user-space daemons and relying on essential kernel subsystems [4, 14] for internal time keeping, network timestamping and accessing PTP hardware clocks.

While PTP can establish a common time base among network nodes (ideally on the same LAN with low-latency links), it is not suited for acquiring a global time reference or for synchronizing geographically or topologically distant networks. For this service, the RIAPS platform relies on the Global Positioning System (GPS) [6] or—as a fall-back mechanism—on NTP. These services are required only at the master nodes in the network and are supported by well-established software packages: chrony [2] or ntpd [9] with the help of the GPSd [12] protocol daemon and relying on the PPS subsystem of the Linux kernel.

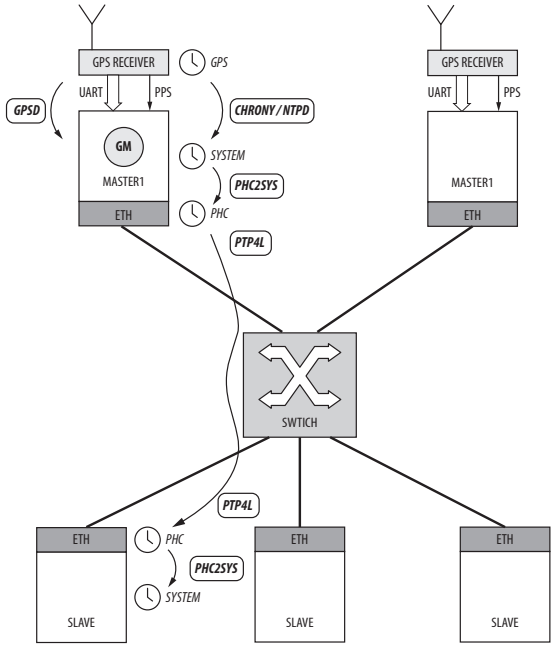

Figure 1: Time-coordination across several clock domains in RIAPS with GPS-disciplined masters and PTP slaves

3 RIAPS TIME COORDINATION ARCHITECTURE

The network architecture and the various clock domains of a RIAPS cluster are shown in Figure 1. Between each pair of clock domains, dedicated Linux services are responsible for monitoring and minimizing the time offsets. The figure shows the corresponding service processes and the direction of the synchronization chain. The various clock domains are:

- **GPS time:** The global, absolute GPS time acquired by a GPS receiver on the master node. The GPS receiver periodically transmits the absolute time values and generates a PPS (GPIO) event at the same moment. The kernel PPS subsystem measures the GPIO pulse using the system clock, which is compared with the value from the receiver to estimate and minimize the time offset.
- **Master system clock:** the wall time maintained by the kernel (`CLOCK_REALTIME`) on the master. This is being adjusted to minimize the difference from GPS time.
- **Master PTP Hardware Clock (PHC)**: an independent clock reference implemented inside the network interface for accurate hardware-base timestamping of network messages. The kernel PHC clock subsystem can measure and provide the offset between the system clock and PHC. On the master node, the PHC is being adjusted to minimize this difference.
- **Slave PHC**: the same independent clock embedded in the network interface on the slave node(s). It is the goal of the PTP protocol to align the slave PHCs to that on the master node.

- **Slave system clock:** the kernel-maintained time on the slaves. All actions on the slave (task scheduling, sensing, actuation) are based on this reference, thus the goal of the entire synchronization chain is to establish tight synchronicity among these clocks. Note, that the direction of synchronization is reversed on the slave: the system clock is being aligned to the slave's PHC.

In each cluster, multiple potential master nodes (nodes with GPS receivers) may exist. At any given time only one master is selected—becomes a grand master—by a built-in agreement protocol in the PTP stack. The remaining potential masters behave like slaves until one of them is elected as the new grand master, in case the current grand master node fails.

4 IMPLEMENTATION

The time synchronization chain described in previous sections depends on the coordinated actions of several hardware / software components. In the RIAPS platform, we used several open source COTS solutions and extended them with custom components and high-level management services, which are (see Figure 2):

- **ChronoCape** is our custom hardware solution for providing a GPS time reference and implementing additional instrumentation tasks as described later
- **GPSd** is a standalone project and service daemon for establishing the connection to the external GPS receiver and implementing various GPS receiver protocols
- **chrony** implements a robust time synchronization service for aligning the master clock to either GPS or NTP references
- **phc2sys** establishes the synchronization between the PHC and system clocks – the actual synchronization direction depends on the role of the node
- **ptp4l** implements the PTP network protocol and aligns the PHC clocks on the slaves
- **tsman** is our custom management tool for configuring and monitoring all other components specific to the current operational role of the node

The `GPSd` daemon is configured to read and process UBX protocol messages [17] through the UART interface on the master node. The results are provided via an IPC shared memory buffer and contains time-pair values for the UART (NTP0 buffer). The first element of the pair is the system time when the last event (UART message) arrived. The second element is the time value received from the other clock domain (GPS time in the UBX message). Note that the first element is unreliable with UART (arrival time of the UART message).

The `chrony` daemon is configured to work with this shared memory segment (NTP0) and also process the PPS (`/dev/pps0`) interface. The first element of the PPS time pairs (system time when the event happened) and the second element of the NTP0 record are needed to do time synchronization. It monitors and combines these pairs as described above and controls/tunes the system clock to minimize the difference between the two clock domains.

The `phc2sys` process only relies on the kernel PHC subsystem, which can provide system time - PHC time pairs. Based on the current role of the node (master or slave) this daemon adjusts one of these clock domains. `ptp4l` depends on another kernel service: hardware-based timestamping of network packets. Using the transmit and receive timestamps, this service aligns the PHC on the slave nodes. Both programs are part of the LinuxPTP project.

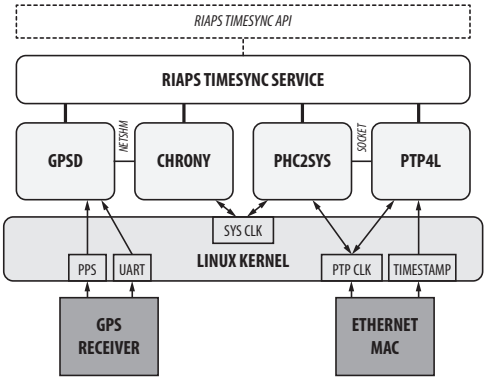

Figure 2: Software components for implementing multi-step clock synchronization on the master node(s)

Instead of running all these processes under the control of a *master integration daemon*, we decided to rely on the more robust and proven built-in service execution mechanisms (upstart [16] or systemd [15]). However, the set of active services and their actual configuration depends on the assigned role of the node. Thus, we implemented a lightweight utility, called `tsman` (Time Synchronization MANager), which ensures that all services are configured and executed properly on the node. The tool can write and verify the configuration files, enables, disables, stops, and (re)starts services when being invoked. The necessary information for these tasks are described in an extensible configuration database. The following roles (profiles) are currently supported:

- **gps-master**: the node is in a dedicated PTP master node with a GPS reference, if no GPS signal is available it can fall back to an NTP source. Requires either a locally attached ChronoCape board or internet connection.
- **ntp-master**: a dedicated PTP master with external NTP reference. Requires internet connection.
- **slave**: the node is forced into a PTP slave role. Requires a PTP master on the LAN segment.

The ChronoCape board (Figures 3 and 4) was designed for providing a highly accurate GPS reference clock and various instrumentation services specifically for RIAPS applications running on the BeagleBone Black hardware platform[7]. It hosts an ARM Cortex-M4F microcontroller and a u-blox LEA-6T GPS receiver module. The microcontroller acts as a bridge for the various communication interfaces: an external USB connector, UART interfaces to the GPS and BeagleBone along with I2C, GPIO and analog connections. It also supports a wired PPS reference with an on-board clock buffer where GPS is not feasible (e.g. indoor applications). For demonstration purposes, the add-on board is capable for generating precisely timed stereo audio outputs via a TRS (audio jack) connector.

The GPS UART interface is typically bridged to the BBB for forwarding all UBX messages to and from the receiver module. The

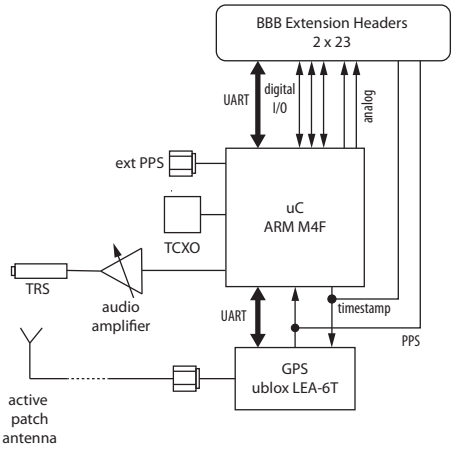

Figure 3: Functional model of the ChronoCape board supporting time synchronization and evaluation

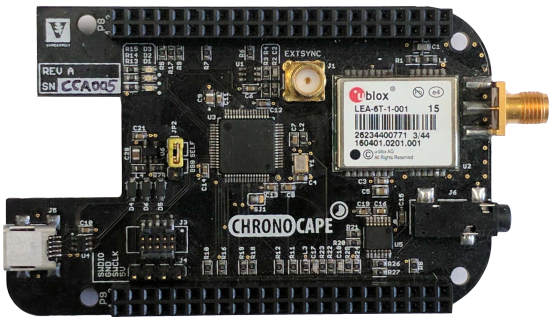

Figure 4: Manufactured and tested prototype of the ChronoCape time reference and instrumentation add-on board for the BeagleBone Black platform

timing on this interface is not critical: the GPS module also provides a PPS signal, which is directly connected to a GPIO pin of the BBB. This GPIO pin is registered as a PPS input, thus the Linux kernel automatically attaches a PPS (`pps-gpio`) driver to this. The UART interface is processed by the `gpsd` daemon, while the PPS values are captured by `chrony`.

Our secondary goal with the ChronoCape board is to build a reliable instrumentation and validation platform for evaluating the performance of the time synchronization service and of the real-time I/O interfaces of the nodes. The board is capable for observing and recording the actions (GPIO and external communication buses) taken by the node on a global timescale. Instrumentation services—implemented as firmware on the MCU—can also generate events for the nodes (pin changes, communication messages, and analog values) at well defined and accurate time instants. For configuring the instrumentation tasks and for collecting data, we can use the external USB connection—completely bypassing the BBB node—or these can be sent over the UART communication link with the node. Thus, RIAPS applications can collect live real-time performance data using an independent and globally synchronized clock domain.

5 EVALUATION

As described in Section 4, there are several clock domains in the distributed time synchronization architecture, with dedicated services—using alternative timestamping technologies and protocols—establishing pairwise synchronization in a cascade from the root reference clock (e.g. GPS) to the leaf nodes (i.e. system clock on slave nodes). Thus, the performance of the time synchronization can be characterized at different points of the architecture. As a system-level metric, the timing accuracy of the slaves—both relative to each other and to an absolute time reference—is one of the most obvious choices.

In this section, we follow a systematic approach for characterizing the performance and major factors of our architecture. Therefore, we will evaluate the following steps:

- **GPS to master node system clock**: (a) by capturing GPS PPS events using the system clock and (b) analyzing the time offsets values reported by `chrony`.
- **Master node system clock to master node PTP hardware clock (PHC)**: by analyzing the time offset values reported by `phcsys` on the master node
- **Master node PHC to slave node(s) PHC**: by analyzing the time offset values reported by `ptp4l` on the slave node(s)
- **Slave node PHC to slave node(s) system clock**: by analyzing the time offset values reported by `phcsys` on the salve node(s)
- **Slave node(s) system clocks**: this is the system-level end-to-end evaluation step by generating GPIO events on multiple slaves at well-defined time instants and measuring these events with a global reference clock

Note that most of the intermediate steps analyze the time offset values between adjacent clock domains as these are observed by the corresponding time synchronization tool—the ultimate goal of these software programs is to minimize the mean and variance of the offsets by carefully adjusting one of the clock domains. Systematic errors—especially in the timestamping approach—remain hidden in these analyses. However, the final system-level characterization provides objective performance results directly applicable to real-world application scenarios.

5.1 Master node: GPS → system clock

This synchronization step is evaluated by observing the system clock timestamps on the external PPS signal generated by a GPS. Differences between consecutive timestamps can tell if the system clock frequency is properly adjusted to the global GPS reference: the mean value of these differences shows the frequency error (should be as close to 1.0 as possible), while the variance can show problems with the PPS signal measurement and/or erroneously jumping system clock. Note that this evaluation approach does not detect if we have a constant offset bias between the GPS and system clock domain. This will be checked by integration-level tests later in this section. We used the `ppstest` [11] and trivial post-processing for this step. Based on a 100,000 point measurement run (more than 24 hours), the average difference between consecutive timestamps is 1.0 seconds (using double floating point calculations). The standard deviation is 13.5 microseconds. The actual distribution is shown in Figure 5. Note, that two symmetric sidelobes and a small number of

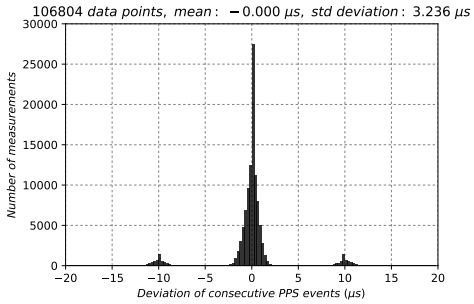

Figure 5: Deviation of consecutive PPS timestamps in master's system clock

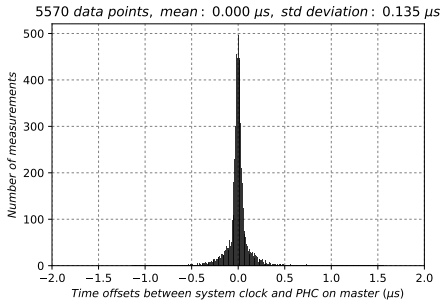

Figure 7: Time-offsets reported by `phc2sys` on the master node

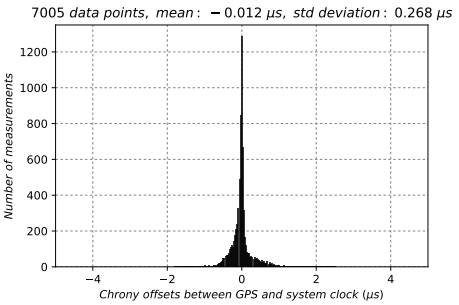

Figure 6: Time-offsets reported by `chrony` after filtering and regression.

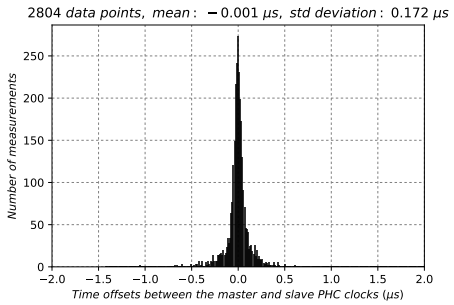

Figure 8: Time-offsets reported by `ptp4l` on the slave node

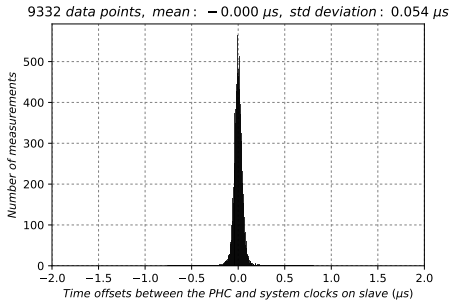

Figure 9: Time-offsets reported by `ptp4l` on the slave node

extreme outliers are the cause for the high RMS value. The built-in median filter and regression model in `chrony` can easily take care of these occasional outliers.

To verify our statement on outlier rejection, we analyzed the same measurement run as reported by `chrony`. The distribution of GPS vs. system time offsets is shown in Figure 6 with a much improved RMS error of 0.268 microsecond. Note that due to filtering and regression, there are significantly fewer aggregated measurement points.

5.2 Master node: system clock → PHC

For characterizing the performance of the next step on the master node, we analyzed the time offsets between the system clock and the PTP hardware clock. The `phc2sys` daemon periodically (every second) measures and logs these offsets and adjusts the PHC clock for minimizing the values. The offset measurement is implemented in the Linux kernel by rapidly triple polling the two clock domains. The results—based on 5500 data points—are shown in Figure 7 and demonstrate a very tight (sub-microsecond) synchronization accuracy.

5.3 Master PHC → slave PHC

This synchronization step is executed across the network using the PTP protocol. The `ptp4l` daemon implements all elements of the protocol both on master and slave nodes. The results—estimated time offsets between the master and the slave PHC clock— can be monitored on each slave node. The results of the analysis of such a log file with 2800 data points are shown in Figure 8. In the measurement setup, the master and slave nodes are connected via a Fast Ethernet switch, which resulted in sub-microsecond variations in the offsets.

5.4 Slave node: PHC → system clock

The `phc2sys` daemon is used again in a reverse role for aligning the system clock to the PHC on each slave. Again, accurate timestamping is provided by the kernel for this step. We use the same output log and post-processing logic for evaluating the performance of this last hop in the time synchronization chain. The results— shown in Figure 9—with 9300 data points are not surprising.

5.5 System-level performance: multiple slave system clocks

All analysis results above showed microsecond-level accuracy at each step in the time synchronization chain. However, the most important and truly relevant performance metric is the time synchronization accuracy across multiple slave nodes—as mentioned before, systematic measurement errors may remain hidden in the previous analysis steps.

For this validation step, we developed a custom application for the slaves. After the time synchronization is established on all nodes, the program starts to generate GPIO pulses based on the slave's own system time (`CLOCK_REALTIME`) at well defined time instants. These pulses are measured by dedicated standalone GPS receivers and the measured timestamps are evaluated. We are interested in the relative differences between the pulses and their alignment to the global (GPS) timescale. Some additional considerations:

- Pulses are generated at every second but using a 500 ms phase instead of the integer second boundary to minimize the interference with the GPS timesync PPS pulse on the master node
- The pulse generation process runs with `SCHED_FIFO` policy and a high (90) real-time priority to minimize jitter
- The pulse generation process uses direct memory mapped I/O operations to minimize jitter and latency on the GPIO pin
- The pulse generation process uses a guard interval before the deadline (500 ms phase boundary): it sleeps for a shorter interval and uses a busy-loop for hitting the exact time instant when the pulse is supposed to be generated

Relative accuracy: The results of taking pairwise differences between the actuation event timestamps on two slave nodes are shown in Figure 10. Although we took several steps to minimize the jitter on the actuation signal, single glitches / hiccups still happen. Therefore, we used a median filter with a kernel size of 3 data points to remove these single / intermittent errors (these are not indicative to the performance of timesync).

Absolute accuracy: The final validation step is to evaluate all GPIO events—generated by multiple slaves concurrently—on the absolute GPS timescale. **The results (see Figure 11) are surprising: all actuation signals have a significant (**$\sim 195\mu s$**) bias.** This common, systematic bias remain hidden in all previous steps and suggests that all nodes in the network have a deterministic offset from the GPS absolute time, most probably introduced by the master node.

Effect of `RT_PREEMPT`: After evaluating several alternative scenarios and investigating the PPS subsystem of the Linux kernel, we identified the root cause of this deterministic bias. The RIAPS platform runs on a fully preemptable kernel using the `RT_PREEMPT` [10] patch for achieving minimal jitter in critical user-space processes. One of the most important changes introduced by `RT_PREEMPT` is to convert regular interrupt handlers into preemptible kernel threads. This means, that servicing most interrupts—such as the timestamping of a PPS GPIO pulse—requires a full context switch in the kernel. The deterministic offset is indeed the context switching time on the selected platform. Note that there is another potential pitfall of using soft interrupt threads for timestamping: any user space process with a higher real-time priority than the kernel interrupt task can introduce a significant jitter in the PPS subsystem. This finding nicely resonates with common wisdom of carefully considering what *"real-time"* really means in different aspects of a system.

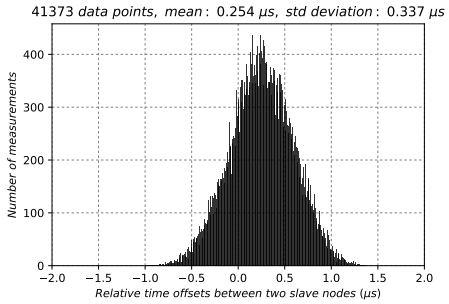

Figure 10: Relative slave-to-slave system-level timing accuracy of GPIO actuation signals

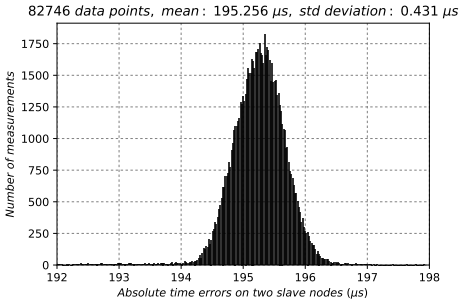

Figure 11: Absolute system-level timing accuracy of GPIO actuation signals on multiple nodes

6 APPLICATION EXAMPLE

To demonstrate time synchronization and its potential uses within the RIAPS platform, we created a sample application to estimate phase angle and predict zero-crossings of a single-phase power transmission line. Suppose an intelligent power grid determines that multiple switches must open, and the switches are optimally opened at a zero-crossing. Each intelligent switch could find a local device providing phase information, then use this information to calculate when a future zero-crossing will occur so as to safely open the switch. The success of this application is predicated upon highly accurate time synchronization between the phase measurement node and the switch controllers.

To demonstrate this use case, our RIAPS application runs on four local BeagleBone Blacks that are synchronized by PTP and offer GPIO and analog outputs. One node must act as both the AC signal generator—called the Signal Generator (Figure 12). The Signal Generator calculates the amplitude of a 60 Hz sinusoid every 1 ms, then writes that value to an analog output (PWM) and publishes said amplitude to the local network, alongside a Linux timestamp of the local system time when that amplitude was calculated. All other

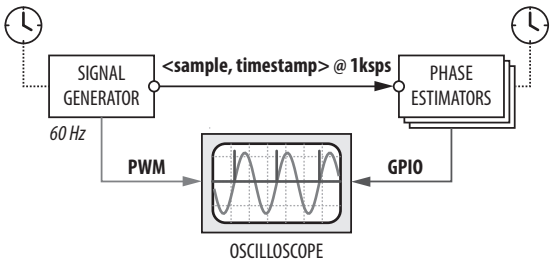

Figure 12: Phase estimation and zero crossing prediction

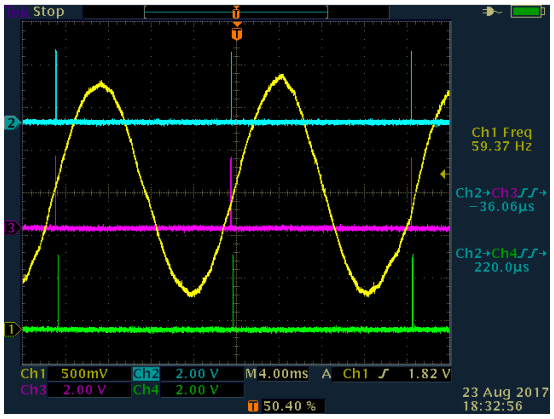

Figure 13: Oscilloscope trace of the analog sine wave (Signal Generator - yellow) with pulses at the estimated zero crossings (Phase Estimators - magenta, cyan, green)

nodes are called Phase Estimators (Figure 12). The Phase Estimators subscribe to the amplitude-timestamp pairs being published by the Signal Generator. The Phase Estimators track these values and react to a positive zero-crossing in the following way; first, each Phase Estimator linearly interpolates between the last two values it read to precisely calculate when zero was crossed. Then, it calculates from its local system clock when the next positive zero-crossing will be and sets a timer for the remaining delay. When that delay timer expires, the Phase Estimator creates a single GPIO pulse, representing a distribution switch opening.

To evaluate this application, both the analog output from the Signal Generator and the GPIO pulses from the Phase Estimators are viewed through a quad-channel oscilloscope (Figure 13). Ideally, the oscilloscope would display a sinusoid with all GPIO pulses—three Phase Estimator timed actions—are aligned with the positive zero-crossings.

Note, that the described application is only a demonstrative use-case of the time synchronization service not a benchmark application for evaluating its performance. Timing bias and jitter have many other sources in this application—e.g. PWM and GPIO peripheral access, task scheduling, event propagation and messaging. Although, these are all important factors for evaluating the system level performance of the RIAPS platform, they are beyond the focus of the current paper.

7 SUMMARY AND FUTURE WORK

We have presented an approach for high-precision time synchronization for a network of small embedded devices. The approach is based on an accurate clock source (a GPS receiver) on, at least, one of the network nodes, the Precision Time Protocol for clock distribution, and a local daemon process that adjusts the system clock of the nodes. Such a synchronized time base allows the precisely time-triggered activation of tasks that is often required in distributed, real-time embedded systems. Experimental results indicate that we were able to achieve high accuracy with some very low-cost solutions. We have also demonstrated the use of the service in a specific application, similar to ones needed for Smart Grids.

The approach is fault tolerant (due to the properties of the PTP) but its performance under fault conditions need to be verified. Additionally, new API-s are needed, possibly implemented in the OS kernel, that allow the very precise scheduling of time-triggered tasks. An additional useful feature would be to notify applications about the current (estimated) accuracy of the local clock.

ACKNOWLEDGMENTS

The information, data, or work presented herein was funded in part by the Advanced Research Projects Agency-Energy (ARPA-E) U.S. Department of Energy, under Award Number DE-AR0000666. The views and opinions of authors expressed herein do not necessarily state or reflect those of the US Government.

REFERENCES

[1] Felipe Cerqueira and Bjrn Brandenburg. 2013. A Comparison of Scheduling Latency in Linux, PREEMPT-RT, and LITMUS RT. In *Proceedings of OSPERT 2013 9th Annual Workshop on Operating Systems Platforms for Embedded Real-Time Applications*, Vol. 1. Paris, France.
[2] Chrony 2017. A versatile implementation of the Network Time Protocol (NTP). (2017). Retrieved June 14, 2017 from https://chrony.tuxfamily.org/
[3] Cyclictest 2017. A high resolution process scheduling test program. (2017). Retrieved May 2, 2017 from https://rt.wiki.kernel.org/index.php/Cyclictest
[4] Thomas Gleixner and Douglas Niehaus. 2006. Hrtimers and Beyond: Transforming the Linux Time Subsystems. In *Proceedings of the Linux Symposium*, Vol. 1. Ottawa, Ontario, Canada.
[5] LinuxPTP 2017. The Linux PTP Project. (2017). Retrieved June 14, 2017 from http://linuxptp.sourceforge.net
[6] P. Misra and P. Enge. 2006. *Global Positioning System: Signals, Measurements, and Performance* (2nd edition ed.). Ganga-Jamuna Press, Lincoln MA.
[7] Derek Molloy. 2014. *Exploring BeagleBone: Tools and Techniques for Building with Embedded Linux*. Wiley. http://www.exploringbeaglebone.com/
[8] T. Neagoe, V. Cristea, and L. Banica. 2006. NTP versus PTP in Com puter Networks Clock Synchronization. In *2006 IEEE International Symposium on Industrial Electronics*, Vol. 1. 317–362. https://doi.org/10.1109/ISIE.2006.295613
[9] NTP [n. d.]. ([n. d.]).
[10] Daniel Bristot Oliveira and Romulo Silva Oliveira. 2016. Timing Analysis of the PREEMPT RT Linux Kernel. *Softw. Pract. Exper.* 46, 6 (June 2016), 789–819. https://doi.org/10.1002/spe.2333
[11] pps-tools 2017. User-space tools for LinuxPPS. (2017). Retrieved June 14, 2017 from http://linuxpps.org/
[12] Eric Raymond. 2008. GPSD. In *The Architecture Of Open Source Applications*, Amy Brown and Greg Wilson (Eds.). lulu.com, Chapter 7, 101–112. http://www.aosabook.org/en/
[13] RIAPS Website 2017. RIAPS. (2017). Retrieved June 14, 2017 from http://riaps.isis.vanderbilt.edu/
[14] John Stultz, Nishanth Aravamudan, and Darren Hart. 2005. We Are Not Getting Any Younger: A New Approach to Time and Timers. In *Proceedings of the Linux Symposium*, Vol. 1. Ottawa, Ontario, Canada.
[15] systemd 2017. System and Service Manager. (2017). Retrieved June 14, 2017 from https://www.freedesktop.org/wiki/Software/systemd/
[16] Upstart 2017. Upstart Intro, Cookbook and Best Practises. (2017). Retrieved June 14, 2017 from http://upstart.ubuntu.com/cookbook/
[17] Various Authors. 2013. *u-blox 6 Receiver Description*. Manual GPS.G6-SW-10018-F. u-blox AG.