Time to Measure the Pi

Peter Membrey
Department of Computing
The Hong Kong Polytechnic University
Hunghom, Hong Kong
peter@membrey.hk

Darryl Veitch
School of Computing and Communications
University of Technology Sydney, Australia
Darryl.Veitch@uts.edu.au

Rocky K. C. Chang
Department of Computing
The Hong Kong Polytechnic University
Hunghom, Hong Kong
csrchang@comp.polyu.edu.hk

ABSTRACT

The Raspberry Pi platform is increasingly being used for network measurement due to its low cost, ease of deployment, and ability to run Linux. Timestamps are a critical part of measurement data, yet the suitability of the Pi for timing has not been established. We use reference hardware to characterize the Pi's STC hardware counter, and to evaluate its performance when paired with a low cost yet powerful GPS 'hat'. We find that the platform can support timing adequate for most measurement purposes, but with some caveats.

Categories and Subject Descriptors

C.2.3 [**Computer Communications**]: Network operations—*Network monitoring*

Keywords

Raspberry Pi, GPS hat, network measurement, PPS, clock synchronization, Internet of Things.

1. INTRODUCTION

The Raspberry Pi (or Pi) has become a widely used computing platform due to its low cost, small form factor, and support for the Linux operating system [3]. For network measurement, its ease of deployment has made it an attractive way to scale out monitoring networks. For example both CAIDA's Ark platform [1], and the BISmark project [2], increasingly employ Pi-based nodes, and Pi's equipped with GPS are now also receiving attention [8].

Although relatively powerful, the processing, memory and networking limitations of the Pi are well recognized [12, 13]. These naturally place constraints on the measurement regimes a Pi-based infrastructure can support, but the range of applications is nonetheless wide, particularly since Pi's or other IoT devices could be used primarily for data collection, with data analysis occurring elsewhere. There is however a fundamental issue that relates directly to the collection platform. Much measurement data involves, if not literally consists of, timestamps. How suitable then is the Pi as a platform for reliable timing? Underlying hardware factors such as low cost oscillators, small thermal mass, low resolution counters, and inexpensive networking architecture, will all impact on its final timekeeping and timestamping performance.

In this paper we examine the suitability of the Pi platform for timing. Our main goals are as follows.

(1) Characterization of the *System Timer Counter*. The STC, which ticks nominally at a rate of 1Mhz, is the Pi's only readily accessible hardware counter. It is therefore the basis of any software clock one could define on the platform (including the system clock under the Raspbian operating system). Clock performance will therefore be limited by the STC's stability and resolution. We use specialist hardware and careful methodology to measure this stability accurately for the first time for each of the Pi-1(B), Pi-2, and the newly released Pi-3.

An alternative counter of higher resolution is that underlying the CPU, however it is not readily accessible under Raspbian, nor used by it (see [15] for access notes). We believe this is because its nominal rate is unreliable, rendering it unsuitable for timekeeping.

The remaining goals concern the situation where Pulse Per Second (PPS) input is available. The main example is when the PPS is provided by a GPS receiver, where the pulses fall on the seconds of *Coordinated Universal Time* (UTC). These pulses are used by system clocks to lock to UTC. Our study of 'Pi+PPS' is motivated by the fact that low cost GPS boards are now available for the Pi, opening up the possibility of using it as a low cost precision timing platform with particular benefits for measurement, in particular where accurate absolute

IMC 2016, November 14-16, 2016, Santa Monica, CA, USA

DOI: http://dx.doi.org/10.1145/2987443.2987476

time is essential. These include studies based on latency comparisons, such as latency benchmarking and mapping, reliable event ordering (needed for example in time-aware distributed databases [9]), or the health checking of public timing infrastructure [21, 20].

(2) Characterization of a low cost GPS *hat*.
We examine a high quality *uBLOX M8Q* GNSS module with a custom fit to the Pi, and compare its PPS output to a GPS synchronized atomic clock reference.

(3) Examination of latency pitfalls in PPS triggering.
The availability of PPS to a network of Pi measurement points enables their use as precise distributed triggers. This would increase the accuracy and resolution of applications such as Internet coordinate systems, and moreover can be achieved independently of potentially inaccurate system clocks. We report on some potential pitfalls of this idea.

(4) Performance of Pi+hat timing.
By comparing 'identical twin' Pi+PPS systems in parallel we report on the expected best case underlying performance of the Pi+hat platform.

It is beyond the scope of this paper to systematically evaluate all elements impacting on the end performance of Pi-based timing. For example we do not evaluate the latency profiles of network hardware, nor evaluate in detail the reaction of *ntpd* (the default system clock synchronization algorithm) to failures in a PPS, a key problem in opportunistic 'antenna out the window' deployments. Nonetheless, in the conclusion we describe the implications of our findings in terms of the end performance of both the Pi+hat and Pi platforms.

Our work provides a strong foundation for future studies. In particular our focus on the availability of PPS is not only about a desire to evaluate a Pi+hat platform. It is the basis of our methodology for the accurate measurement of the goals above, which can also be used and/or adapted for the benchmarking of alternative platforms, or future counters on the Pi which may become available. For example it may be possible to synthesize a reliable counter of high resolution by combining the STC and the CPU counter in a similar way to Xen Clocksource [6]. Our methodology also allows for the benchmarking of methods to estimate counter stability when PPS is *not* available. This can be done by exploiting RADclock [19], and will also form the object of our future work.

2. TESTBED HARDWARE

Our testbed consists of a number of Pi's, six units of a particular GPS hat, GPS receivers (more accurately, GNSS, using both GPS and GLONASS satellite constellations), an atomic clock, and additional measurement equipment.

2.1 The Pi and its Hat

We work with all three generations of the Raspberry Pi, including the recently released Pi-3. Although there have been some variations within these (we use Pi-1(B)), they mainly affect CPU, RAM and additional features such as wireless networking, bluetooth capabilities and the like, rather than the core hardware including the STC hardware counter that we focus on here. The Pi's all have *General Purpose Input Output (GPIO)* pins, which can be used to input a PPS signal.

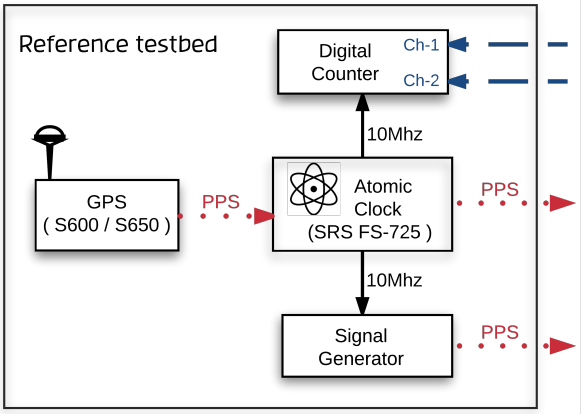

Figure 1: The reference testbed equipment.

To provide the GPS functionality, a GPS module from UPUTRONICS (previously HAB Supplies) was used. This device was chosen since, despite being a low cost board (£34.99 at the time of writing) and designed to physically fit (like a hat) to the Pi, it uses a very high quality uBLOX M8Q GPS receiver which features a PPS output enabling high precision locking to UTC. Cheaper GPSes rely solely on messages sent using the NMEA protocol [14], which do not arrive with the same precision as pulse edges. The chipset supports various modes to improve its reliability for time synchronization, such as limiting the number of NMEA messages it sends, and allowing *Stationary mode*: telling the receiver it is physically stationary reduces the degrees of freedom and so improves the time fix. Finally, the M8Q model provides a TCXO (Temperature Compensated Crystal Oscillator), which has a higher stability than of a standard crystal, though still below that of an atomic clock. This cost to performance ratio makes it an ideal board for developing larger IoT devices.

2.2 Reference Testbed

Benchmarking a GPS system requires an accurate and robust reference against which to compare. To create such a reference, industry standard equipment was chosen from leading manufacturers, widely known and respected in the industry.

We cascade three devices to produce, finally, a PPS output which is extremely stable both at small and long time-scales, and which offered flexible configuration for our needs. This cascade, shown in Figure 1, consists of: GPS receiver → atomic clock → signal generator.

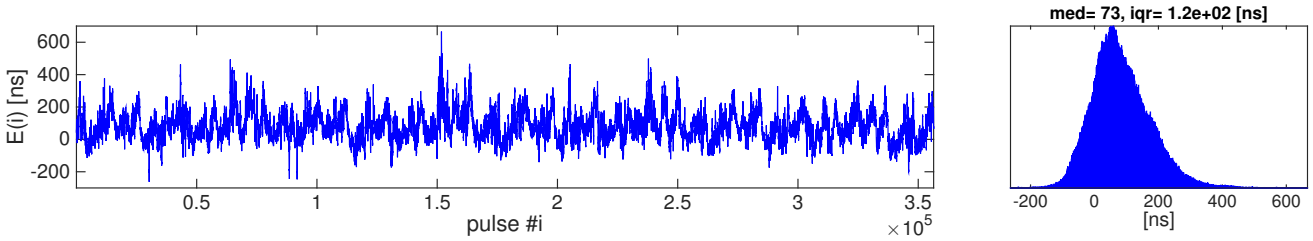

Figure 2: Pulse errors $E(i)$ of the hat over 99 hours. Left: timeseries, Right: histogram of all values.

The first device is a Microsemi S600 GPS Network Time Server, which provides a UTC synchronized PPS which is highly stable on long time-scales.[1] To improve stability over shorter time-scales, and to provide holdover protection in case of satellite reception problems, we use the GPS's PPS output to discipline a SRS (Stanford Research Systems) FS-725 Rubidium Desktop Reference. This atomic clock produces a reference PPS which will converge to a more stable form of the output of the S600. Finally, because the $1.2\,V$ output level of the FS-725's PPS is potentially too low for the Pi to reliably detect (especially when daisy chaining and testing a number of Pi's in parallel), we use the 10Mhz output of the FS-725 to precisely discipline a Tektronix AGF1022 signal generator, whose own output can be easily configured to generate a PPS of our choosing. This PPS source was used for experiments with the Pi.

The Pi's GPIO interrupt trigger level is usually around $1.8\,V$, but can range from $0.8\,V$ to $2.0\,V$ [5]. Whilst the Pi-2 and Pi-3 triggered reliably at $1.2\,V$, the Pi-1 did not. We set the signal generator's output to $2\,V$, which allowed reliable triggering for all Pi's.

Each GPS receiver (both the Microsemi references and the hats) has it own antenna, which was placed outside a nearby window. The number of satellites visible during experiments was monitored. It was well above the minimum required for each receiver to maintain a reliable and accurate time fix. For example the S600 saw typically 6 and 8 satellites from the GPS (L1) constellation.

2.3 Measurement Equipment

In addition to our PPS reference, a Rigol DS1104Z Oscilloscope and a Keysight 53230A digital frequency counter were used for performing visualization, comparison, and measurement of pulse trains.

The counter was disciplined (in hardware via phase locked loop) by the 10Mhz output of the FS-725. The counter's time-interval measurements are estimated to have an accuracy of 50 ps (1 sigma) for our signals. The oscilloscope does not support external synchronization, however it contains a TCXO and was only used for comparing signals over sub-μs intervals, where errors due to the TCXO drift would be in the pico-second range.

[1]In some cases a higher end Microsemi S650 was used instead of the S600. The improvement is negligible for our purposes. For simplicity we refer to the S600 below.

3. CHARACTERIZING THE HAT'S PPS

We begin with the analysis of the hat performance (Goal 2), as this is the logical first step, being independent of the Pi. We characterize the pulse train of the hat in three ways: via per-pulse errors, stability analysis, and pulse shape.

3.1 Experiment Design

To measure per-pulse errors, we feed the pulse trains from the hat and our atomic clock reference into separate channels of the digital counter. The counter is set to trigger on each hat pulse and return the time interval to the next reference pulse.

We offset (via a hat configuration option) the hat pulses by $\delta = -10\mu$s (to sub-ns error) to ensure they always arrive before their corresponding reference pulses. This offset is removed in post-processing.

The resulting data is the error in the arrival time $t_h(i)$ of the i-th hat pulse:

$$E(i) = t_h(i) - t_r(i),\ i = 1, 2, \ldots$$

compared to the reference $t_r(i)$.

Since (by necessity) our reference is assumed perfect, we know that $t_r(i) = t_r(1) - 1 + i$. We can therefore recover the hat pulse times as

$$t_h(i) = t_r(i) + E(i) = t_r(1) - 1 + i + E(i)$$

and obtain the Inter-Pulse Time sequence, used in stability analysis, as

$$\mathrm{IPT}(i) = t_h(i+1) - t_h(i) = 1 + E(i+1) - E(i).$$

To examine pulse shapes we employ the oscilloscope, collecting summary statistics on rise and fall times, and pulse widths for the two pulse trains separately. The scope was set to trigger at $1.2\,V$ for the SRS and $0.86\,V$ for the hats (50% of the recorded output voltage at 50 ohms). The trigger level, together with the choice of edge, in fact defines where pulses 'are'. The rise/fall times can therefore be read as bounds on variations in pulse position arising from alternative trigger choices that may be found in equipment accepting PPS input.

We collect data from our 'main' hat in an experiment 99 hours long, and also from four other hats of the same type, each 12 hours long.

3.2 Results

Errors: The error histogram shown in Figure 2 for the main hat is well behaved. With a median error of 116ns and an inter-quartile range of only 73ns, by itself this would suggest that the hat is a more than adequate hardware basis upon which to build a software clock with sub-μs precision. The other hats are similar, with median errors of $[-83, -35, -81, -74]$ ns, corresponding inter-quartile ranges of $[105, 134, 119, 121]$ ns, and ranges all under 1μs (recall range= max−min). However, the time series in Figure 2 shows that the errors $E(i)$ are temporally correlated, with oscillatory behavior on multiple timescales, motivating a deeper stability analysis.

Stability: The IPT(i) sequence contains the variations in pulse rate about the ideal of a perfect 1PPS. Understanding how this rate fluctuates over time reveals and quantifies the impact of diverse influences that operate at different timescales.

It is traditional to characterize the variability or *stability* of clock oscillators via the *Allan Variance* [11]. This can be interpreted as a robust measure of the variance of rate, averaged over timescale τ, as a function of τ (see [18] for a more detailed discussion). Simple time domain approaches such as the empirical autocorrelation function are unsuitable as they are highly biased on non-stationary data. The Allan Variance is in fact equivalent to a Haar wavelet energy spectrum plot, and therefore enjoys a number of statistical robustness features of wavelet analysis (see [4]).

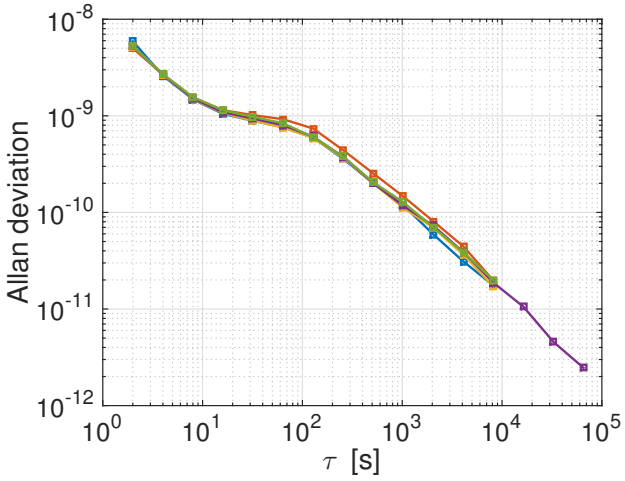

Figure 3: Allan Deviation of IPT(i) as a function of timescale τ.

The *Allan Deviation* (square root of the Allan Variance) as a function of τ for IPT(i) is shown in the log-log plot of Figure 3 for all five hats (the main hat from Figure 2 stands out as the longer curve).

The results in each case were very similar. As expected, the value drops as the averaging interval τ grows. Even at the smallest scale, here $\tau = 2$s, the relative rate error/deviation is under 6×10^{-9}. (At small scales the sawtooth on the PPS input is being averaged out. The slope past $\tau = 100$ s is -1, indicating white phase noise.)

This plot is used below to put into context the Allan Deviation of the Pi's hardware counter, the STC.

The above analyses were performed on IPT(i) sequences from which any outliers were replaced by neutral surrogate values. This is essential to avoid significant distortion of the Allan Deviation, which is sensitive to impulses. An effective surrogate is to replicate a neighbor: we set IPT(i) = IPT$(i-1)$ for each outlier i. In our data we encountered only a single such outlier, with $E(i) \approx 0.9s$, due to a counter triggering error. If any such outliers were to reach the Pi, it is critical that they can be effectively dealt with by the upstream software clock algorithm or significant errors can result. We return to this general point in the conclusion.

Metric	min	1st	median	99th	max
rise time [ns]	2.8	2.8	3.3	4.0	4.1
fall time [ns]	3.7	4.1	5.1	6.2	6.5
pulse width [ms]	100	100	100	100	100

Table 1: Pulse shape statistics for the hat.

Metric	min	1st	median	99th	max
rise time [ns]	5.1	5.2	5.6	6.4	6.6
fall time [ns]	2.6	2.6	2.8	3.5	3.9
width [ns]	9900	9950	10000	10050	10050

Table 2: Pulse shape statistics for the SRS.

Pulse Shape: Tables 1 and 2 give results obtained from 1,000 pulses from the hat and our reference respectively. In both cases the pulses are extremely consistent in shape, with tight rise and fall times of comparable size. The pulse width of the hat is much wider than that of the reference (so wide it results in limited measurement resolution), but this is not important when rising edges are used in triggering, as here.

4. STABLITY OF THE STC

We now address Goal 1, the stability of the STC on the Pi. As the core system hardware varies over the Pi-1, Pi-2 and Pi-3, we provide results for each.

4.1 Experiment Design

We would, ideally, like to read the STC (we call this *raw* timestamping) at precisely periodic timepoints with small period, however performing this within an operating system is challenging. There are three issues, all of which concern latency: timestamp triggering, timestamping location, and counter reading.

We deal with counter reading by accessing the counter directly via memory mapping. This provides a minimal access latency uniformly across kernel and user-space. Our approach to triggering is to exploit the almost perfectly periodic pulses from our reference PPS. Finally, to

minimize the latency between the delivery of pulses to the GPIO pin and their timestamping, the timestamp location is chosen to be at the beginning of the IRQ routine responding to pulse interrupts. Kernel modifications were added to pass the raw timestamps up to a data logging application.

The resulting data are the raw timestamps $\mathrm{STC}(t_i)$, where t_i is the true time at which the i-th pulse was timestamped. We denote the corresponding raw inter-pulse values by

$$C(i) = \mathrm{STC}(t_{i+1}) - \mathrm{STC}(t_i).$$

The timeseries $C(i)$ combines variability from three sources: (i) the underlying pulses; (ii) the latency between pulse arrival and timestamping; (iii) the variability of the STC itself. It is not possible to separate these within the Pi, which is why it is so important to send in the reference quality pulse train to minimize the first component, and to make every effort to minimize the second as described above, so that $C(i)$ is a low noise measurement of the underlying STC behavior. It is difficult to provide an accurate figure for the error bound for the error in our methodology, that is of latency source (ii) above, as it depends on hardware characteristics which can be difficult to obtain. We believe it however to be typically under 1μs.

We conduct a 20-day long experiment where each Pi is fed a copy of the same reference pulse from the signal generator. Hence the pulse and temperature conditions are identical for each.

4.2 Results

Because the STC is a free running counter, the timeseries $\mathrm{STC}(t_i)$ displays drift, and so, unlike the case of the hat's PPS, $C(i)$ is not centered precisely around its expected value (namely 10^6, since the STC is nominally a 1MHz counter). The actual average period of $C(i)$ will vary from board to board and can (and must) be measured, together with the stability about that value.

For each of the Pi's the variation in $C(i)$ is small: the inter-quartile range is exactly 1μs in each case, a reflection of the limited resolution of the counter. Accordingly, we use a robust mean rather than the median to determine the 'central' value of the counter period. We find the STC period for the Pi-1, Pi-2 and Pi-3 to differ from the nominal 1μs by 19.3, -6.76, and -8.13 parts per million (PPM) respectively. Here and below missing pulses, and other rare anomalies/outliers in $C(i)$, have been replaced by neutral surrogates, again to avoid distorting the Allan Deviation.

The Allan Deviation plots for each Pi are shown in Figure 4. At small scales up to 100s the results are ordered as one might expect according to generation, with the older Pi-1 having the most variability and the Pi-3 the least. At intermediate scales between 100s and 1000s temperature effects enter in, which can affect different platforms differently (this is more apparent when looking at subsets of the full trace, which have some

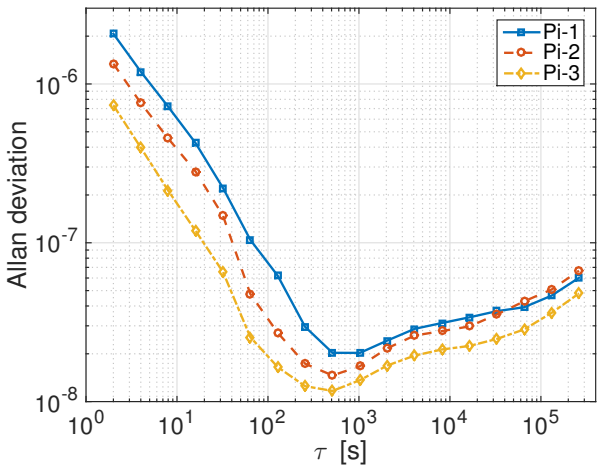

Figure 4: Allan Deviation of $STC(i)$ as a function of timescale τ for each generation of Pi.

overlap over this scale range. The full traces are fortuitiously well nested here). Finally, at scales beyond 1000s the variability actually starts to increase as diurnal temperature profiles rather than hardware or software effects dominate, and so the platforms become increasingly similar.

The Allan Deviation plot for Pi-3 is very similar in general terms to that seen in our prior characterization of commodity PCs and servers [19, 17, 7]. The methodology here however, being based on PPS triggering, is superior, resulting in lower spurious variability. It follows that the STC is in fact more variable than the counters previously studied, as one might expect from a smaller platform, even more so for the Pi-2 and Pi-1. What is important however is that in each case two key characterizations: (1) beyond short timescales the variability drops below 10^{-7} and stays there, and (2), the location of the minimum point being of the order of $\tau = 1000$s, agrees with that of our prior work. This indicates that the Pi's *are suitable for timekeeping from the stability point of view*, implying underlying limits to final clock quality which are not much worse than those for PC platforms (see [19, 17]).

The accuracy and reliability of a final clock solution however, depends not only on the underlying counter hardware, but also on system latencies elsewhere in the system, and how successfully a synchronization algorithm can filter these to create a software clock which is not just accurate, but robust to anomalous events.

5. PITFALLS IN PPS TRIGGERS

Precise coordinated triggering can increase the accuracy of distributed measurement applications such as latency-based Internet coordinate systems, by allowing precise relative ranging of multiple sources to landmark nodes [10]. Triggering based on synchronized PPS has the important advantage that it can be virtually independent of host system clocks, in particular of their

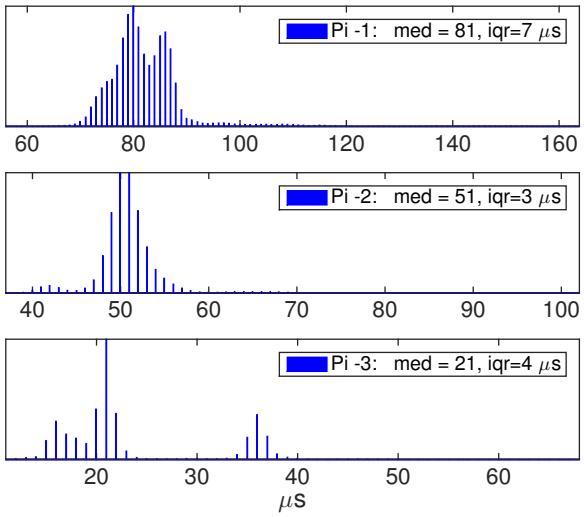

Figure 5: Kernel to user-space latency $L(i)$.

errors and potentially limited resolution. The low cost of Pi+Hat nodes makes it feasible to assemble a substantial measurement network with this capability. Its potential could however be derailed by latency issues, in particular those due to crossing the user and kernel space divide. We examine this question here.

5.1 Experiment Design

Linux provides a standard way to access a PPS signal via the Kernel PPS interface. We wrote a user-space data logging application that uses this call to block on each pulse. Upon each callback we immediately read the STC using memory mapping, and also retrieve and log our IRQ-based STC timestamp. For each generation of Pi we collected several hours of data, including periods of minimal load, and high load induced by file transfers.

5.2 Results

We first examined the Inter-Pulse values $C(i)$ taken at each timestamping location separately. Take for example results for the Pi-2. Although having identical median values, they exhibited markedly different variabilities, with [kernel, user] standard deviations of $[3.1, 418]\mu$s, and ranges of $[0.11, 32.5]$ms. Note that this is despite the benefit of the low latency memory-mapped STC reads.

A more direct approach is to examine the latency $L(i) = \text{STC}(t_i^u) - \text{STC}(t_i^k)$ between the raw timestamps read at time t_i^k in the kernel and t_i^u in user-space. The histogram of $L(i)$ is given in Figure 5 for each Pi generation, showing in each case values from the minimum to the 99.9-th percentile. Though a clear generational improvement is seen, the gap between the arrival of the pulse and its receipt in user-space can be significant, indeed outliers continue out into the ms range.

The experiment also collected instances where the pulse was seen by the kernel, but the callback failed. This would represent a critical failure to a measurement infrastructure intending to exploit PPS availability as a means to coordinate distributed measurement.

In conclusion, although dedicated support exists to access PPS from user space, for both reliability (failed callbacks) and accuracy (user-space latency) reasons a more direct kernel based raw timestamping is needed to ensure the hat's potential for well under 1μs triggering accuracy, exhibited in Figure 2, is realized. This is particularly true for the special application of disciplining the system clock itself.

6. PERFORMANCE OF THE PI + HAT

In this section we ask an obvious but important question, how good in practical terms is the performance of the Pi+hat compared to a Pi connected to an expensive, reference PPS?

6.1 Experiment Design

We compare twin Pi-3's in a single experiment 28hrs long where everything is identical across the twins: the Pi-3 hardware, software image, the hat and its satellite visibility, experiment duration and temperature environment. The only difference is the source of the PPS: in one it comes from the hat itself, in the other this has been replaced by our signal generator reference PPS.

For the first time we will examine timestamps from the system clock, which is disciplined by the *ntpd* daemon synchronizing to the input PPS in a standard configuration for each twin. Timestamps are made within the IRQ routine responding to pulses (just after our raw timestamping location). We compare the system clock timestamps $S_r(i)$ and $S_h(i)$ for the twin with the reference and hat PPS respectively.

6.2 Results

We examine the Inter-Pulse Time sequence

$$\text{IPT}_x(i) = S_x(i+1) - S_x(i)$$

for each Pi, where x is one of h or r.

Figure 6 gives a representative subset 3000 pulses wide of each IPT_x timeseries, and histograms of all values between the 0.1 and 99.9-th percentiles. The two series are very similar, however the outliers in the hat case are larger than those of the underlying pulses themselves, which have a range under 1μs (Figure 2). This shows that the system clock built on hat pulses can have, overall, similar accuracy to one built on a high end reference system (provided satellite reception is maintained). However, it also highlights the importance of algorithm stability, as small instabilities at the pulse level are being amplified rather than suppressed by *ntpd*, albeit slightly, despite the ideal conditions.

Whereas $\text{IPT}_x(i)$ looks at errors in measuring 1 second intervals, the error of the system clock as it evolves across the entire trace is given by examining $S_x(i) - (S_x(1) + i - 1)$. For each twin we observe a fluctuating

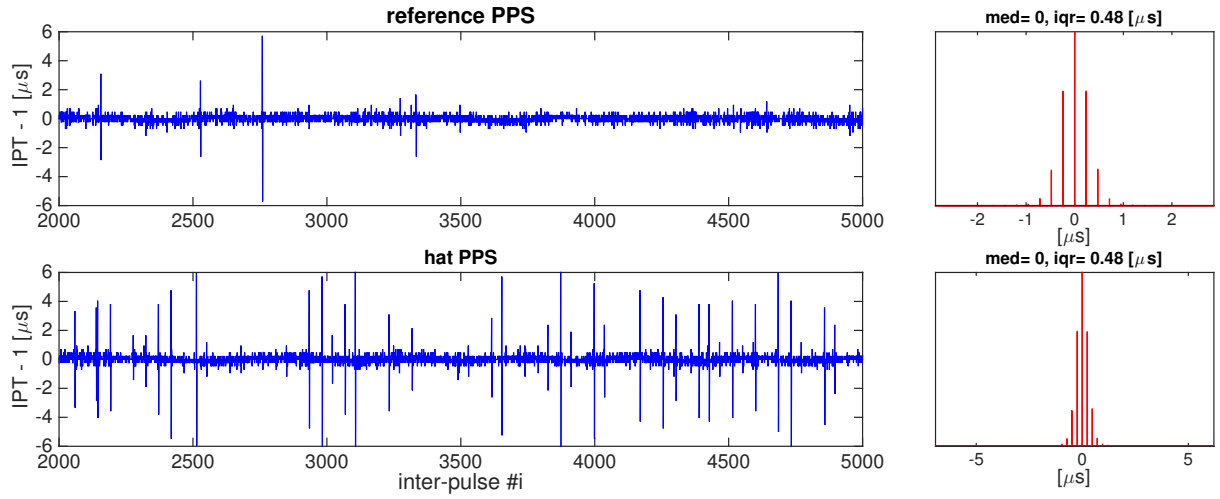

Figure 6: Inter-Pulse Intervals from the twin experiment. Left: timeseries, Right: histogram.

periodic wander in a 30μs band in this quantity, (indicating that the software clock (in)stability is dominating the small differences in pulse quality. For reference, the best case wander on a PC platform is closer to 10μs.

7. CONCLUSION

We conclude by synthesizing our findings into a response to the question, *can we trust the Pi for timing?*

In terms of timing hardware fundamentals, results from Goal 1 showed that the STC's stability was an adequate basis for a software clock, not very different from that of a larger PC platform [17]. The 1μs resolution however is a current limitation that will be inherited by any software clock based on it. As pointed out in the introduction however, this could potentially be overcome.

The results from Goal 2 showed that a well chosen, inexpensive GPS source can be almost indistinguishable from an expensive reference for timing needs below 1μs, however this is, crucially, predicated on a reliable PPS which implies in particular consistent satellite coverage. Moreover, Goals 3 and 4 each point to the need to manage access latencies properly to fully benefit from a PPS. Operating system and other latencies are more significant on a Pi than on a larger platform.

Beyond hardware fundamentals comes synchronization fundamentals, which depend critically on the synchronization algorithm, and its ability to manage latency variability both in the underlying reference, and in timestamps thereof. With PPS available, Goal 4 shows that Pi+hat using *ntpd* on Raspbian can perform to within a small factor of a Stratum-1 server on a larger platform, but erratic behavior if the PPS stability were for some reason degraded cannot be ruled out, particular under high system load. We performed some simple tests of system clock robustness, by repeatedly disabling the hat's pulse for a few seconds at a time and then restoring it. We found that the system clock suffered immediate disruption following each missing pulse event, and took minutes to recover.

For a Pi without PPS, the timestamps used for system clock definition/synchronization would be of packets exchanged with remote timeservers, which is problematic as each Pi has a high latency network interface. For example the Pi-3 has a USB based interface which is polled from the operating system rather than interrupt driven. Under ideal conditions latencies lie in the 10's of μs range, but can easily reach 100's of μs or even the ms level under heavy load. It is well documented that *ntpd* has significant stability issues [19, 17, 16]. When network and system latency variability reach certain 'tipping points', stability is lost, and errors can jump from a best case value of the order of 1ms to a wandering error with an amplitude of 10's or 100's of ms, or even beyond. Because latencies are higher on a Pi, it is more vulnerable to this occuring. Whether any given error is acceptable is entirely application dependent, however reliability is always important, as without it error bounds cannot be placed on measurements.

To give an idea of the best possible performance of the Pi without a PPS based on *ntpd*, we performed an additional 30 hour experiment where a Pi-3 was a Stratum-2 client, synchronizing over the LAN to a Stratum-1 reference NTP server (the SRS) with no other network traffic or system load, while system clock timestamps were still collected based on in-kernel PPS triggers as per Section 6. The $\text{IPT}_h(i)$ timeseries now exhibited spikes in the $\pm 20\mu$s range and periodicity with amplitude 5μs. More importantly, the accumulated error $S_x(i) - (S_x(1) + i - 1)$ exhibited a wander in a 370μs band, compared to 30μs previously. This is approaching the ms level, even under these very unrealistic ideal conditions.

Feedforward synchronization algorithms such as RADclock [19] do not suffer from the feedback instabilities

above and can deliver high accuracy (wander under 1ms under reasonable assumptions). Moreover, because RADclock treats time difference measurement differently from absolute time at a fundamental level, it can provide PPS-like accuracy for quantities such as round-trip and inter-packet times, and delay variation, without the need for PPS [19]. This is ideal for measurement efforts, such as CAIDA's Internet topology mapping, which are based on RTTs. Of course, individual packet timestamps will still suffer from the latencies of the network hardware, even with a perfect system clock.

To conclude, timing sufficient for most network measurement purposes is at a fundamental level possible on a Pi, but to achieve it may require replacing *ntpd* to ensure accuracy, reliability, and meaningful error bounds. With the right clock synchronization approach, the limitation to timestamp accuracy on the platform is likely to be the latency of the network interface.

Acknowledgment

Partially supported by Australian Research Council's Linkage Projects funding scheme #LP120100073, in partnership with Symmetricom (now Microsemi), an ITSP Tier-2 project grant (ref. no. GHP/027/11) from the Innovation Technology Fund in Hong Kong, and a research grant from the Joint Universities Computer Center of Hong Kong (ref. no. H-ZL17). Thanks also to Lewis Masters, Nick Gustafson, Brendan Horan, and Michael Wouters for their insights, and the reviewers and our shepherd for helpful comments.

8. REFERENCES

[1] Archipelago monitor locations. http://www.caida.org/projects/ark/locations/.

[2] BISmark. http://www.projectbismark.net.

[3] Raspberry Pi. https://www.raspberrypi.org/.

[4] P. Abry, D. Veitch, and P. Flandrin. Long-range dependence: revisiting aggregation with wavelets. *Journal of Time Series Analysis (Bernoulli Society)*, 19(3):253–266, May 1998.

[5] ARM. RealView Platform Baseboard for ARM1176JZF- S User Guide., 2011.

[6] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the Art of Virtualization. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177, New York, NY, USA, 2003. ACM.

[7] T. Broomhead, J. Ridoux, and D. Veitch. Counter Availability and Characteristics for Feed-forward Based Synchronization. In *Int. IEEE ISPCS'09*, pages 29–34, Brescia, Italy, Oct. 12-16 2009.

[8] R. Calvo-Palomino, D. Giustiniano, and V. Lenders. Collaborative Signal Monitoring and Decoding with Low-Cost Software-Defined Radio. In *Proc. ACM SIGCOMM Internet Measurement Conf.*, Santa Monica, Nov. 14-16 2016.

[9] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, W. Hsieh, S. Kanthak, E. Kogan, H. Li, A. Lloyd, S. Melnik, D. Mwaura, D. Nagle, S. Quinlan, R. Rao, L. Rolig, Y. Saito, M. Szymaniak, C. Taylor, R. Wang, and D. Woodford. Spanner: Google's globally-distributed database. In *Proceedings of the 10th USENIX*, OSDI'12, pages 251–264, Berkeley, CA, USA, 2012.

[10] E. K. Lua, T. Griffin, M. Pias, H. Zheng, and J. Crowcroft. On the accuracy of embeddings for internet coordinate systems. In *IMC'05: Proc. ACM Internet Measurement Conference*, pages 11–11, Berkeley, CA, USA, 2005.

[11] D. L. Mills. The Network Computer as Precision Timekeeper. In *Proc. Precision Time and Time Interval (PTTI) Applications and Planning Meeting*, pages 96–108, Reston VA, Dec. 1996.

[12] R. Mok, W. Li, and R. Chang. Improving the packet send-time accuracy in embedded devices. In *Proc. PAM*, 2015.

[13] R. Mok, W. Li, R. Chang, K.-W. Yung, C.-H. Chan, and Y.-S. Poon. An automated testbed for profiling the packet send-time accuracy of embedded devices. In *Proc. TRIDENTCOM (Poster Session)*, 2015.

[14] NMEA. Publications and Standards from the National Marine Electronics Association (NMEA) / NMEA 0183, Nov. 2008.

[15] J. Regehr. High-Resolution Timing on the Raspberry Pi. http://blog.regehr.org/archives/794.

[16] J. Ridoux and D. Veitch. Ten Microseconds Over LAN, for Free (Extended). *IEEE Trans. Instrumentation and Measurement (TIM)*, 58(6):1841–1848, June 2009.

[17] J. Ridoux, D. Veitch, and T. Broomhead. The Case for Feed-Forward Clock Synchronization. *IEEE/ACM Transactions on Networking*, 20(1):231–242, Feb. 2012.

[18] D. Veitch, S. Babu, and A. Pásztor. Robust Synchronization of Software Clocks Across the Internet. In *Proc. ACM SIGCOMM Internet Measurement Conf.*, pages 219–232, Taormina, Italy, Oct. 2004.

[19] D. Veitch, J. Ridoux, and S. B. Korada. Robust Synchronization of Absolute and Difference Clocks over Networks. *IEEE/ACM Transactions on Networking*, 17(2):417–430, April 2009.

[20] D. Veitch and K. Vijayalayan. Network Timing and the 2015 Leap Second. In *Proc. of PAM 2016*, Heraklion, Crete, Greece, March 31 - April 1 2016.

[21] K. Vijayalayan and D. Veitch. Rot at the Roots? Examining Public Timing Infrastructure. In *Proc. of IEEE INFOCOM 2016*, San Francisco, CA, USA, April 10-15 2016.