Performance analysis of Statime as compared to LinuxPTP

Dion Dokter Ruben Nijveld David Venhoek

April 8, 2022

1 Introduction

In this paper, we present a quantitative comparison of statime synchronization performance as compared to the LinuxPTP implementation of PTP. In particular, we focus on comparing the precision and offset of the time synchronization of both setups.

The comparison was done over relatively short time intervals of about two hours. Over these periods we find that, due to incomplete support for hardware timestamping, the precision of statime is still 2 orders of magnitude larger than that of LinuxPTP.

2 Measurement setup

Our measurement setup consists of 3 main parts:

- An endrun ninja ptp grandmaster clock synchronized via GPS.
- A raspberry pi 4 compute module with an Intel i210 NIC
- A Basys-3 FPGA development board setup to measure pps pulses.

The ptp grandmaster is connected via a direct ethernet cable to the i210 nic, and this connection is used to carry the PTP messages. Both the ptp grandmaster and the intel nic are configured to produce pulse-per-second outputs. Furthermore, the raspberry pi is also configured to produce a pulse-per-second output on its gpio pins.

All these pulses are measured by the FPGA relative to its internal clock. Although this internal clock is not precise, we can estimate its frequency using the pulse-per-second signal from the ptp grandmaster.

The grandmaster configuration is the default provided by endrun, and the grandmaster is given time to fully lock onto the GPS signal before start of measurement (specifically, we wait until the clock quality reported by gpsstat is at level 3).

On the raspberry pi, depending on the run we either start the LinuxPTP stack or statime. We then wait with starting the measurement until the self-reported offset to the grandmaster appears stable.

3 Error estimates

In our analysis we have included the following 4 main error sources: Cable length differences, fpga input path differences, discretization error, and fpga clock instability. Below we will discuss each, including their contributions to our estimated error.

3.1 Cable length

The setup contains cables of multiple lengths and varying electrical characteristics. We will estimate these errors pessimistically and assume maximum systematic errors up to the propagation delay of our longest possible cable. As all our cables are below 1.5 meters in length, this gives us an estimated 15ns of systematic error.

3.2 FPGA input architecture

Although care has been taken in the VHDL code the fpga has been programmed with to keep the input path for all of the signals identical, due to the fact that these interact with different pieces of the physical hardware there can still be offsets between processing delays for the inputs. Based on the documentation of the FPGA's architecture, we assume these to contribute 1 fpga clock cycle of systematic error.

3.3 Discretization error

The pulse per second signals arrive at the fpga unsynchronized with the fpga's clock. Due to this, their measured arrival time may be off by up to 1 clock cycle, giving us a statistical error of 1 fpga clock cycle.

3.4 FPGA clock instability

The clock in the fpga is of relatively low quality, and may therefore be somewhat unstable. We will use the pulse-per-second signal measured from the grandmaster clock to estimate the frequency instability of the fpga's clock, which will provide us with an additional statistical error component.

4 FPGA clock callibration

We callibrate the FPGA clock using the data from the GM pulse-per-second signal. This process purely uses difference between arrival times of these pulses, hence errors from cable length can be ignored in the following analysis. Note that over short time intervals, the clock frequency seems quite stable, with the second interval only varying by 1 clock cycle around a central value over short time periods. To compensate for any longer-term drift of the FPGA clock frequency, we do this calibration on a per-run basis.

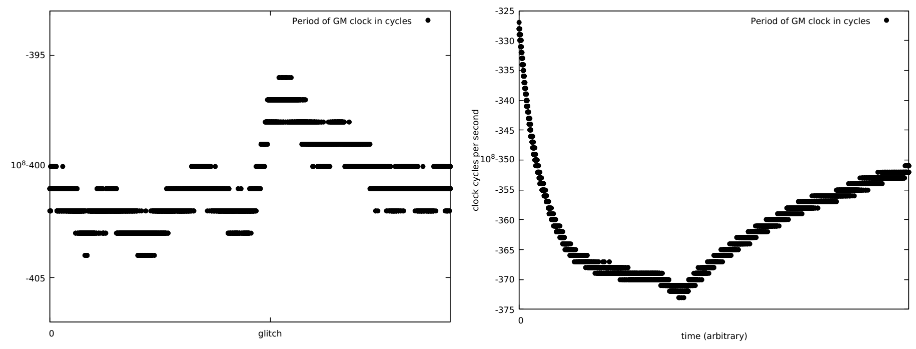

Figure 1: FPGA clock cycles per second over time, time axis is arbitrary. On the left is data from the LinuxPTP run, on the right from the statime run. Glitch marks points where some seconds of data have been removed due to jiggling of connectors causing spurious edges.

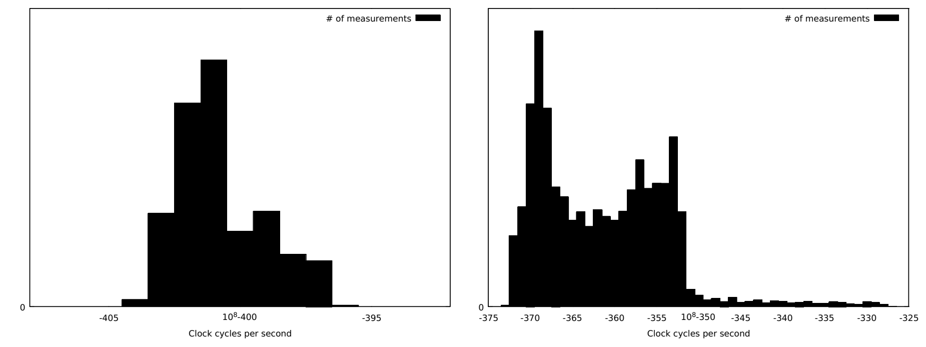

Figure 2: FPGA clock cycles per second as measured using the GM pulse-per-second output. On the left is data from the LinuxPTP run, on the right from the statime run.

For the LinuxPTP run, the measurement data are plotted in Figure 2. Analysing this shows the fpga clock ticks 99999599$\pm$1.7(stat.)$\pm$1.0(sys.) per second. Hence, a single clock tick of the fpga clock takes 10.00004010$\pm$0.00000017(stat.)$\pm$ 0.00000010(sys.) nanoseconds during the LinuxPTP run.

For the statime run, analysis shows the fpga clock ticks 99999638$\pm$7.8(stat.)$\pm$ 1.0(sys.) per second. Hence, a single clock tick of the fpga clock takes 10.00003620$\pm$ 0.00000078(stat.) $\pm$ 0.00000010(sys.) nanoseconds during the statime run.

5 LinuxPTP performance analysis

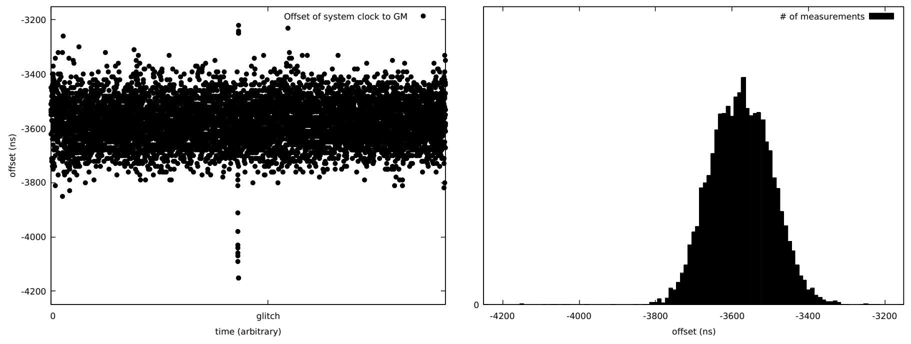

Figure 3: System clock offset to GM, both overtime and frequency of observations, as synchronized by LinuxPTP. (Negative values indicate that the clock is ahead of the GM.)

Analysing the offset data for the system clock synchronized by LinuxPTP, we find that this gives a constant offset $-3576,40 \pm 0.95(\text{stat.}) \pm 25(\text{sys.})$ nanoseconds, which could be eliminated by tuning the port assymetry. Furthermore, we see 90% of all observations fall within $126 \pm 10(\text{stat.})$ nanoseconds of this constant offset.

6 Statime performance analysis

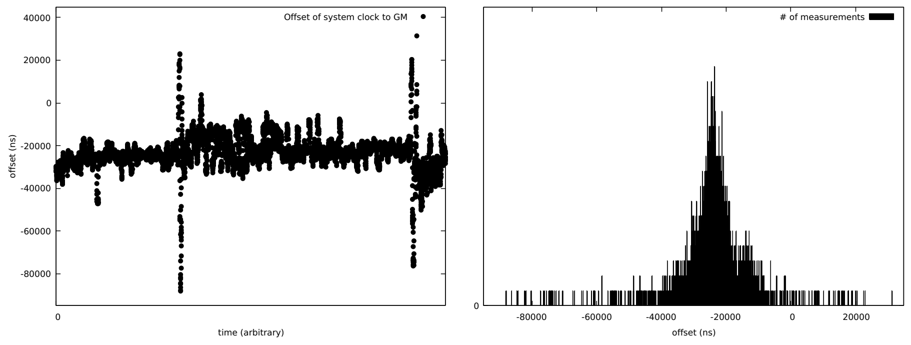

Figure 4: System clock offset to GM, both overtime and frequency of observations, as synchronized by Statime. (Negative values indicate that the clock is ahead of the GM.)

Analysing the offset data for the system clock synchronized by Statime, we find that this gives a constant offset $-23570 \pm 90(\text{stat.}) \pm 25(\text{sys.})$ nanoseconds, which could be eliminated by tuning the port assymetry. Furthermore, we see 90% of all observations fall within 11250 ± 10(stat.) nanoseconds of this constant offset.

7 Conclusions

Comparing the results of the statime and LinuxPTP, we observe that statime is 2 orders of magnitude less precise. We think that this is primarily the result of the fact that statime does not yet have support for hardware timestamping.

In line with the observed decreased precision, we also see an associated increase in fixed offset. This is likely primarily due to assymetry within the linux kernel's networking stack, which becomes more significant when relying on software timestamping.