Estimating noise for clock-synchronizing Kalman filters

David Venhoek
Tweede Golf B.V.
Nijmegen, The Netherlands
david@tweedegolf.com

Estimating noise for clock-synchronizing Kalman filters

David Venhoek
Tweede Golf B.V.
Nijmegen, The Netherlands
david@tweedegolf.com

***Abstract*—We implemented methods to determine the main uncertainty parameters for a Kalman-based clock servo whilst running that servo. This allowed us to provide a clock servo that can provide strong synchronization performance and accurate synchronization error estimates without manual tuning for any specific hardware. Evaluating the servo on real hardware we find that it provides synchronization accurate to about twice the resolution of the clock. Furthermore, the parameter estimation seemed to introduce only a modest amount of additional startup time for the filter at around 10 minutes.**

***Index Terms*—Clocks, Synchronization, Kalman filters, Parameter estimation**

I. Introduction

When distributing time over a network, a critical component is processing of the measurements and using them to decide how to steer the clock. The logic for how to do this is often called a clock servo.

With recent wider scale deployments in datacenters, such as those at Meta [1], how to design a well-functioning clock servo for use with PTP [2] remains an interesting topic. Proportional-integral control-based servos such as those considered in [3] work but require tuning specific to the used hardware and network conditions. Furthermore, such servos cannot provide an estimate of current synchronization precision.

Kalman filter [4] based clock servos such as those in [5]–[8] can provide such estimates. However, they are still limited by the need for estimates of both measurement noise and oscillator stability. Furthermore, those estimates have direct impact on the precision estimate of the synchronization. Thus these will need to be provided on a per-setup basis to provide valid precision estimates.

This requirement for setup-specific calibration imposes an implementation barrier for end users, especially in the context of general purpose software implementations of PTP. They either need specialized knowledge to properly tune the implementation for their hardware, or accept worse synchronization performance and/or precision estimates.

This paper aims to alleviate this, by providing methods for determining the primary noise parameters needed for a Kalman filter based clock servo whilst running the servo. This allows the servo to effectively "learn" the properties of the environment in which it is running, removing the need for the user to manually tune it.

Part of the work in this paper was supported by the Sovereign Tech Fund.

We describe the design of our servo in section II, zooming in on the estimation of the uncertainty inputs to the Kalman filter in section III. This servo was implemented and tested on real hardware, as described in section IV. Finally, we discuss the results of this testing, and how this approach can be further refined, in sections V and VI

II. Kalman filter

In the design of our servo, we follow an event-based approach similar to that in [8], with an additional state parameter for the transmission delay. Letting θ denote the offset between remote and local clock, ω the frequency error of the local clock with respect to the remote clock, and δ the (one-way) transmission delay. Thus, our state is $s = (\theta, \omega, \delta)^T$, with corresponding evolution matrix

$$F(\Delta) = \begin{pmatrix} 1 & \Delta & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \tag{1}$$

where Δ is the time interval according to the local clock since the filter was last brought up-to-date. The filter is only brought up-to-date to process newly received measurements, meaning that Δ varies with the random intervals on which delay messages are sent.

For our noise model, we take the continuum limit of the model used in [8]. This is extended for the transmission delay by assuming the later evolves independently, adding additional covariance proportional to the square of the transmission delay.

This gives a process covariance matrix for the infinitesimal timesteps that looks like

$$\lim_{\Delta \to 0} \frac{Q(\Delta)}{\Delta} = \begin{pmatrix} B & 0 & 0 \\ 0 & A & 0 \\ 0 & 0 & C\delta^2 \end{pmatrix} \tag{2}$$

which, after integration, yields a process noise covariant matrix that looks as

$$Q(\Delta) = \begin{pmatrix} \frac{A}{3}\Delta^3 + B\Delta & \frac{A}{2}\Delta^2 & 0 \\ \frac{A}{2}\Delta^2 & A\Delta & 0 \\ 0 & 0 & C\Delta\delta^2 \end{pmatrix}, \tag{3}$$

where A, B and C are parameters that we will determine later. Here t_f is the new time of the filter, and t_i the previous. All three represent the covariance of their respective random walk process after one second. Here, A corresponds to the frequency random walk component of the oscillator noise, B

to the phase random walk component of that noise, and C to an assumed random walk noise on the transmission delay. The top-left 2x2 submatrix of this form is effectively the limit $\tau_{KF} \to 0$ of Equation 18 in [8], whilst keeping $N\tau_{KF}$, $\frac{\sigma_\theta^2}{\tau_{KF}}$ and $\frac{\sigma_\gamma^2}{\tau_{KF}}$ constant.

Note that we explicitly let the process noise for the delay depend on the delay δ itself, to make changes in it relative to the total transmission delay. For more detail on the reasoning behind this, see subsection III-C

This matrix $Q(\Delta)$ is used to update the covariance matrix P according to

$$P(t_f) = F(t_f - t_i)P(t_i)F(t_f - t_i)^T + Q(t_f - t_i), \quad (4)$$

A. Measurements

Comparisons of the local and remote clocks are done through the PTP protocol [2], with delay measurements done through the end-to-end delay mechanism. Thus, we get measurements associated with the reception of PTP Sync messages, and measurements associated with the exchange of end-to-end delay messages. For both, we define the measured offset θ_M as the difference of the local timestamp on the message with respect to the remote timestamp on the message.

We treat the sync and delay messages as providing separate measurements, as these are sent independently and can have different rates of occurring. For Sync messages, the difference between the timestamps is predicted to be the sum of the delay and the offset, giving a measurement matrix $M_{\text{sync}} = (1, 0, 1)$, whilst for delay messages, the delay contributes in the opposite direction, giving a measurement matrix $M_{\text{delay}} = (1, 0, -1)$. We assume the noise on both measurements to be identical, giving a measurement noise matrix $R = (D)$ parameterized through a single parameter, which we again will determine later.

These determine the Kalman gain

$$K = PM^T(MPM^T + R)^{-1} \quad (5)$$

where P is the a priori uncertainty, and M can be either M_{sync} or M_{delay} depending on the specific measurement. The Kalman gain in turn determines the update to the filter state

$$s' = s + K(\theta_M - Ms) \quad (6)$$

$$P' = (1 - KM)P \quad (7)$$

where s is the a priori filter state, s' the a posteriori estimate of the state, and P' the a posteriori estimate of the uncertainty.

B. Steering

The resulting filter state after measurement is used to determine how to steer the clock. Above a threshold of 1 millisecond, any offset is corrected immediately with a step. Below this threshold, the clock's frequency is adjusted such that the resulting clock rate will fully compensate the error in 2 PTP Sync intervals, resulting in a gradual slew to the correct time.

These adjustments to the clock are then also incorporated in the clock state. For a step, the offset in the state is compensated to match the new situation. Similarly, when changing the frequency of the clock, the state's estimate for frequency is updated to match the new frequency after correction. We assume that steering does not introduce additional process noise. Since in practice, most modern computer oscillators are not actively steered but rather steering means changing the mapping of a counter, this only neglects a potential rounding error. This rounding error is negligible when compared to the oscillator and network noise and hence can be safely ignored.

III. Estimating the error parameters

A. Measurement noise

The measurement noise is easiest to estimate. For this, we consider pairs of sync and delay messages sent closely together. For such a pair, a roundtrip time can be calculated from the difference between the two timestamps made by the time transmitter, and the two timestamps made by the time receiver on those messages. By selecting for messages that are close together, we minimise any error introduced from the uncertainty on frequency differences between the local and remote clock[1].

This process gives an estimate of twice the transmission delay, independent of the offset between time transmitter and time receiver. Although we are not interested in this estimate itself (the transmission delay is already estimated by the Kalman filter), the variance of these estimates will be twice the variance in the transmission delay. Hence, calculating the standard estimator of this variance provides us with twice the measurement variance, allowing us to dynamically determine the value for the parameter D.

We implement this by keeping a ring buffer with the last 32 roundtrip times measured, calculating the sample variance of this set. Only pairs of sync and delay messages that are at most 200 milliseconds apart are used to measure these roundtrip times. During startup, we won't have a full set of samples. To cope with this, a fixed (large) value is used for the estimate of the measurement noise until we have at least 4 measurements. Then, until we have at least 8 measurements, instead of using a proper variance we use the difference between the largest and smallest roundtrip time instead as the estimate for D. This provides conservative estimates during initial startup, preventing overconfidence of the filter early on.

B. Oscillator noise

Estimating oscillator noise is quite a bit harder, since there is no orthogonal measurement that can be directly used to estimate its noise. However, we can try to infer information about it from the observed differences between the prediction of the filter and the actual measurements we obtain. If these differences are repeatedly smaller than estimated by the Kalman filter, our process noise estimate is likely low, and vice-versa.

To actually apply this idea, we first need to make a simplification, as we can only deal with a single parameter and

[1]Note that we can trivially compensate for any deliberate frequency error due to steering of the local clock.

currently still have 2 (A and B) to determine. We made the choice to set $B = 0$. Although not theoretically justifiable, measurements we have done on the quartz oscillators used in network cards such as the Intel I210 suggest this works well in practice, as over long time intervals the terms proportional to A appear to dominate, and on the short time intervals discretisation noise (which we take to be part of the measurement noise) dominates (See also figure 2).

Next, we need to deal with the fact that measurement noise, especially at smaller sync intervals, may be a significantly larger contribution to the total measurement uncertainty than process noise. We deal with this by running a second Kalman filter in parallel with the primary clock steering filter. At the start of each oscillator noise estimation cycle, this filter is initialized from the state of the primary filter. Then it is run in lockstep with the primary filter, but unlike the primary filter, it only evolves due to passing time, getting corrected for the active steering inputs, but not absorbing any intermediate measurements. This is continued until the second filter's prediction uncertainty is significantly larger than both the starting uncertainty of the filter and the measurement noise.

At this point, the uncertainty estimate of the second Kalman filter is dominated by the value of A. This means that, if the value of A is reasonably close to the physical properties of the oscillator, we would expect this uncertainty to be a reasonable predictor of the difference between the measurement and prediction for the next incoming measurement. At this point, we calculate the probability p of seeing a difference equal to or smaller than the current and start a new noise estimation cycle.

The probability p is calculated by first calculating $X = \frac{(\theta_M - Ms)^2}{MPM^T + D}$, where M is the measurement matrix for the measurement in question (both measurement types are used for this). Assuming all random variables involved in the Kalman filter to be normally distributed (the standard Kalman filter assumption), X has a chi-square distribution with 1 degree of freedom, allowing calculation of p with the inverse cumulative density function of that distribution.

The above process results in a series of probabilities which we use to judge if the current estimate is either low, correct, or high. If we judge it to be low, then the value of A is increased by multiplying it by 4, and if we judge it to be high, we divide it by 4.

To judge whether we are low or high, we keep a counter value. This value starts at 0 every time the value of A is changed. Then, for each probability calculated, we compare it against $1/3$ and $2/3$. If it is at or below $1/3$, we subtract 1 from the counter. If it is at or above $2/3$, we add 1 to the counter. If it is between $1/3$ and $2/3$ we move the counter 1 towards zero (adding or subtracting as needed).

Through this procedure, the counter value provides an indication of the accuracy of the currently used value of A. If close to correct, the counter will stay close to 0. However, if the current estimate for A is low, the counter will slowly grow to a large positive value, and if our estimate for A is high, it will slowly grow to a large negative value.

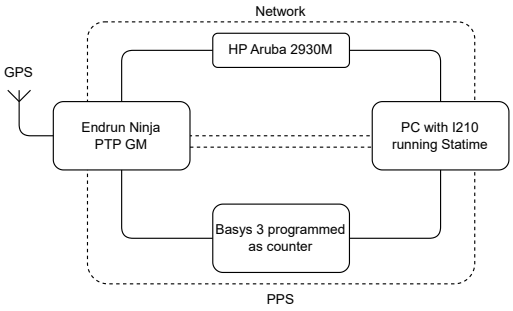

Fig. 1. Diagram showing main components of test setup used to test the filter. Filter was implemented in Statime, run on the PC with an i210 network card.

We use this as our indicator of A needing to be changed by checking the distance of the counter to 0 against a limit (16 in our implementation). Once the counter reaches this limit, if it is positive the value of A is increased, and if it is negative the value of A is decreased as described above.

In the above procedure, the choice for the probability boundaries at $1/3$ and $2/3$, as well as the multiplier and counter cutoff, were essentially arbitrary. The two probabilities were chosen to be symmetric around $1/2$ to ensure no bias in the estimation procedure towards either low or high values, and from experimenting we found that the chosen values work well.

This approach still requires an initial estimate for A. However, because we dynamically update it, a relatively poor estimate can be used, as the above estimation algorithm will ensure it will get to within a reasonable factor of the actual value. The only disadvantage is a longer startup period during which the Kalman filter is not operating at peak performance. We have found 10^{-16} to be a reasonable initial value for A, which is also what was used in the evaluation below.

C. Transmission delay noise

The random walk noise for the transmission delay was primarily incorporated to allow the filter to eventually adapt to changes in network conditions. We chose to make this entry in the process noise matrix explicitly proportional to the current delay. Our motivation for doing this is that a larger transmission delay corresponds to having more cable and equipment between the two nodes, and hence more sources which can change characteristics, likely leading to proportionally larger changes in the actual transmission delay over time.

In the testing below, such network conditions changes were not present, and we haven't evaluated the impact of this parameter beyond ensuring it was small enough to not meaningfully influence errors on the other two parameters. For our testing C was chosen such as to produce a 1 percent uncertainty on the transmission delay after one hour, assuming a starting point of no uncertainty and no further measurements.

IV. Test setup

A complete clock servo consisting of the Kalman filter, associated noise parameter estimation, and clock steering was implemented in the Statime PTP software[2]. For testing, commit 3845563 was used. Statime was installed on a computer with an Intel I210-based network card. This was connected through an HP Aruba 2930M JL319A switch to an Endrun Ninja PTP grandmaster clock. The Statime software and grandmaster clock were configured using the default parameters from the default PTP profile [2], using the end-to-end delay mechanism and Ethernet as the transport.

For characterizing the oscillator on the network card, a programmable pulse output (PPO) of the grandmaster clock was connected to one of the gpio pins on the network card. The grandmaster clock was configured to produce either one or one thousand pulses per second on this programmable pulse output, and the network card was configured to timestamp both edges of these pulses as they arrived using its internal clock.

During this characterization process, the network cards clock was not synchronized. From the resulting timestamps, the rising edges were selected and used to calculate the Allan deviation [9] of the oscillator for timespans between 100 milliseconds and 1000 seconds. The lower pulse rate was used to reduce the amount of measurement data for the large measuring time needed for calculations of the Allan deviation for the larger time intervals.

For measuring the synchronization quality of the servo implementation, a modified setup was used. The grandmaster PPO was configured to produce a pulse-per-second output aligned to the start of the second, as was the gpio pin of the network card. Both were connected to a Basys 3 FPGA programmed to timestamp incoming pulses against an internal (PLL-generated) clock running at 1GHz, using the internal deserialisation (SERDES) blocks in the FPGA. The difference in arrival times was then used to calculate offset between the grandmaster and the network card clock.

This approach was chosen as the FPGA was able to provide a better resolution of 1ns as compared to that of timestamping done by the network card, which has an 8ns resolution. Furthermore, this makes the time difference measurement independent of the hardware under test. The FPGA can be used for this purpose even though its internal clock is of rather poor quality (100ppm rated offset) since measuring offset between the clocks involves measuring small intervals (less than five microseconds), whereas the oscillator characterization would have required larger measurement intervals up to half a second, significantly increasing the impact of the frequency error of the FPGA.

A. Error estimates

For our error analysis, we consider the pulse-per-second signals produced by the grandmaster and network card to represent the ground truth of their respective clocks. This leaves only errors on the measurement by the FPGA, which is predominantly determined by the discretisation error introduced by its clock rate, which is 0.5ns.

[2]https://github.com/pendulum-project/statime

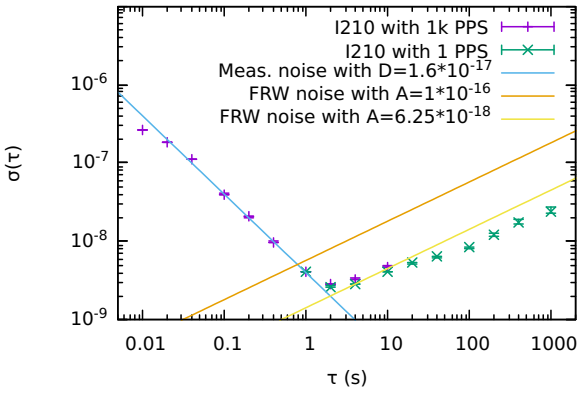

Fig. 2. Allan deviation of an oscillator on an Intel I210 network card. For comparison the theoretical predictions from the noise model for measurement and frequency random walk noise are provided in the solid lines.

Furthermore, due to the specific way the timestamping is implemented, there are effectively 8 sample points that each run on their own 125MHz clock, which have a phase relation such that there is a clock edge on precisely one of these 8 clocks for every edge of the 1GHz clock. Due to electrical differences between these samplers the sampling point of these may not be perfectly spaced over the 8ns interval of the lower-speed clock. To also compensate for the fact that the input signals are uncorrelated to the internal clock, we pessimistically assume the further error introduced by this to be 0.5ns.

For the characterization of the oscillator, the FPGA is not used and instead the pulse-per-second signal is directly timestamped by the target oscillator. This oscillator in the network card has a clock rate of 125MHz, giving an estimate for the discretisation error of 4ns. However, as this is part of the intrinsic properties of the oscillator we are characterizing, we consider this just another noise component we want to characterize and include in the measured Allan deviation.

Systematic errors due to cable lengths and differences in input/output path lengths on both the FPGA as well as the grandmaster and network card are dealt with by focusing primarily on measures of variation, ignoring average offset between the clocks. For such measures, any systematic offsets drop out. This holds both for variation of the offset, as well as for more involved measures such as the Allan deviation of the clock pulses.

The Allan deviation in figures 2 and 6 was calculated using the overlapping estimator from [10], as standardized in [11]. Errors were determined using the noise type giving the worst uncertainty, on a per-point basis. For the measurement using the FPGA, the measurement errors from the FPGA where then added.

With regards to the characterization of the clock on the network card, it must be noted that the room which was used for the measurements was not climate controlled. This lead to

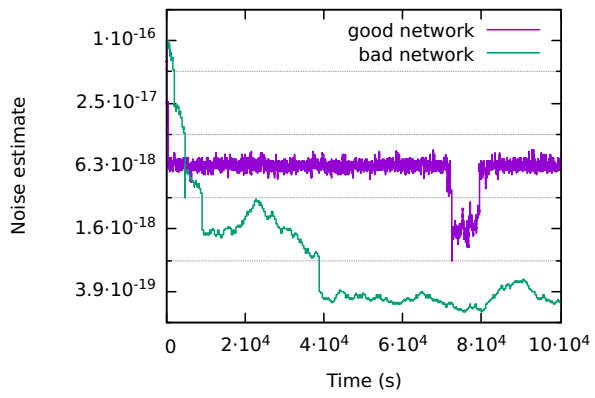

Fig. 3. Estimation state for the frequency random walk noise. Each of the five areas between dashed lines corresponds to one value for the estimated frequency noise. Position within these bins shows the counter value at that time. The two lines correspond to good and bad network conditions respectively.

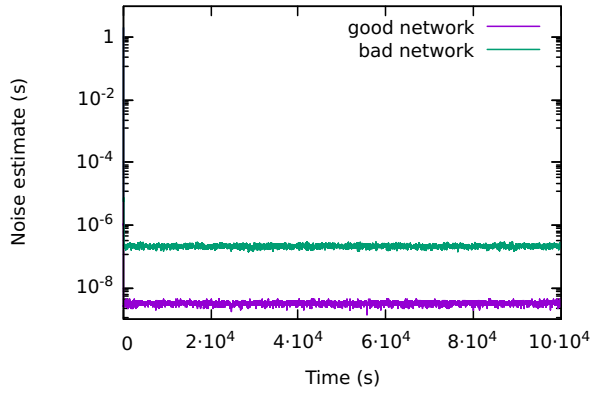

Fig. 4. Estimation state for the measurement error. The two lines correspond to good and bad network conditions respectively.

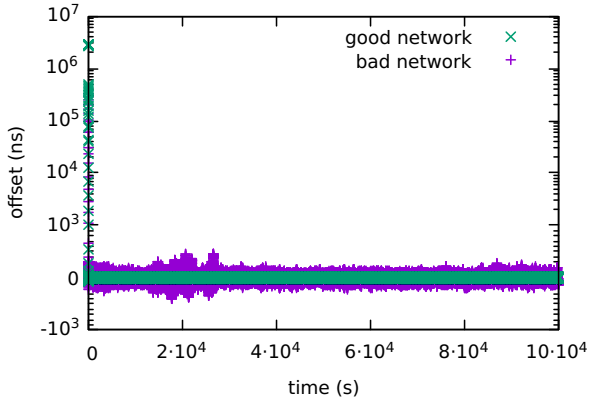

Fig. 5. Measured offset of the network card clock to the grandmaster, relative to the average offset over the measurement period excluding startup. Note that during the first 6 seconds, the software was not yet synchronizing the clock. Scale is linear between -10^3 and 10^3, and logarithmic above that. The two sets of points correspond to good and bad network conditions respectively.

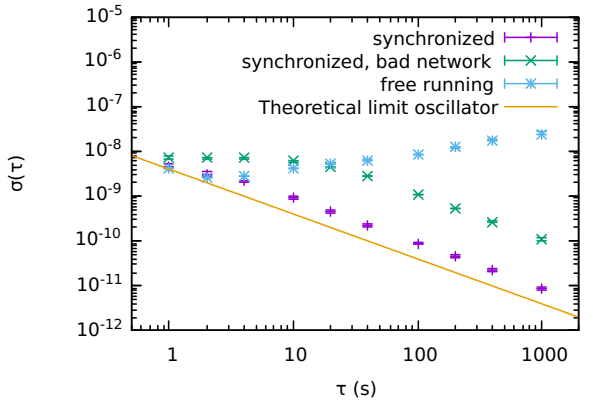

Fig. 6. Allan deviation of the network card clock, as compared to the theoretical limit given the oscillator's 125MHz clock. The two sets of points correspond to good and bad network conditions respectively. The free running data given for comparison is the same as the 1 PPS data from figure 2.

slightly different climate conditions during the characterization of it, which is visible in the overlap region of the results. Although we were unable to quantify this, based on the observed results (see also figure 2) this is likely the dominant uncertainty on this measurement.

The uncontrolled temperature also likely influenced oscillator stability during the testing with the servo. However, we there consider it just part of the operating noise of the oscillator that the servo needs to control for.

V. Measurement results

A. Clock characterization

The clock in the Intel I210 network card was characterized as described above. From the resulting data, we calculated the Allan deviation, the results of which are shown in figure 2. We see two regimes: At low τ we see the resolution of the clock dominating the oscillator error. At higher τ a frequency wander component is the dominant error component.

B. Servo performance

To evaluate the servo performance, we ran Statime on the machine with the I210 network card and observed it for 10^5 seconds. During this period, the Statime software started up, connected to the PTP network and synchronized the network card clock to that of the grandmaster clock. A second test of the same duration was run with an additional consumer switch (DLink Go-SW-5G) on the network path to worsen network measurement error. Both tests used the same parameters. Figures 3, 4 and 5 show the behaviour of the noise estimators and the resulting synchronization behaviour.

Note that the behaviour of the oscillator stability estimator consists of two variables, the actual estimate and the counter used to modify that estimate. As the estimate itself is rather coarse and doesn't change much, we have combined these. The individual bins on the y-axis, separated by the grey lines correspond to the values of the estimate itself. Within the bin, the height of the line is then controlled by the value of

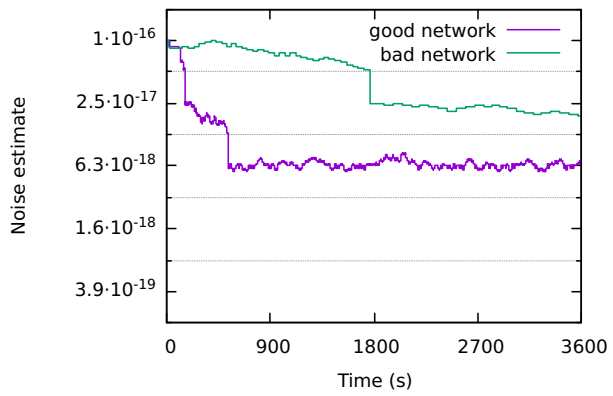

Fig. 7. Estimation state for the random walk noise during startup. Each of the five areas between dashed lines corresponds to one value for the estimated frequency noise. Position within these bins shows the counter value at that time. The two lines correspond to good and bad network conditions respectively.

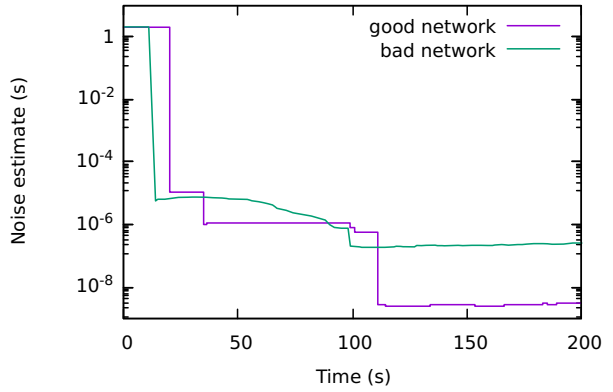

Fig. 8. Estimation state for the measurement error during startup. The two lines correspond to good and bad network conditions respectively.

the counter. This intuitively shows how the counter works to indicate how confident the algorithm is in its current estimate, with higher counter values indicating the current estimate is likely low and vice versa.

We clearly see that the error estimation algorithms converge quite quickly on their estimates, and remain relatively stable afterwards. The oscillator noise estimate takes the longest, and also shows shows a bit more variation after the initial period. The author believes this is likely due to minor changes in climate within the room in which the test was conducted over the duration of the test.

The difference in oscillator noise estimate between the two tests is likely in part also caused by poor climate control. However, a second factor that is likely to play a role here is that, due to the larger network errors, the estimator is measuring the oscillator noise effectively over larger time periods. This, combined with the fact that for large measurement periods, the oscillator noise is not quite frequency random walk noise anymore, likely also results in a smaller estimate with worse network conditions.

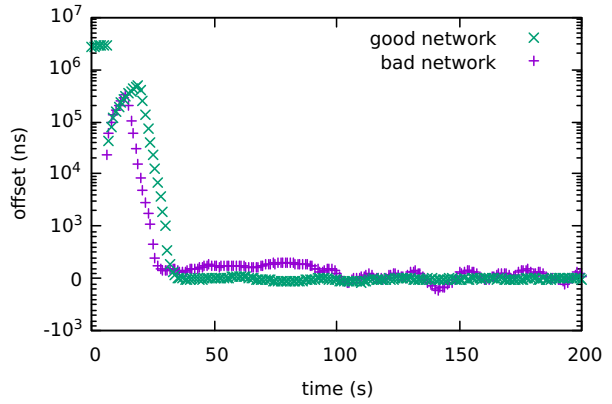

Fig. 9. Measured offset of the network card clock to the grandmaster during startup, relative to the average offset over the measurement period excluding startup. Note that during the first 6 seconds, the software was not yet synchronizing the clock. Scale is linear between -10^3 and 10^3, and logarithmic above that. The two sets of points correspond to good and bad network conditions respectively.

In terms of stability of the offset, we initially see rather marked movement during the startup period. This then rapidly stabilizes, and remains mostly within fixed bands for the remainder of the test. The difference in network conditions is clearly visible in the stability of the synchronization, but is not as big as would perhaps naively have been expected based on the difference in network error. This is due to the filter correctly recognizing it should average more between subsequent measurements.

This better-than-naively expected performance is also visible in the more quantitative analysis in figure 6. Here, we see that the stability only worsens by about a factor of 10, despite the network error increasing by almost a factor 50.

C. Startup behaviour

Let us now zoom in on the servo's behaviour during startup, shown in more detail in figures 7, 8 and 9. First, consistent with the parameters from the chosen PTP profile, it took the implementation about 6 seconds to select the grandmaster clock as its time source. Upon the first measurements immediately afterwards, we see the software synchronizing roughly to the clock, reducing synchronization error to several hundred microseconds.

At this point, we still use a very conservative estimate for the transmission delay, which has the effect of both inhibiting finer synchronization of the clock as well as proper learning of the oscillator noise. As the software gets a more accurate picture of the variations in transmission delay, we see marked improvement in the synchronization quality around the 100 second mark.

Note that naively, one would expect the variance estimate already to be good after 8 round trip time measurements, which is expected to be collected in about 20 seconds. However, the first few estimates happen during a period where the clock frequency is still rather poorly measured, resulting in the first round trip time estimates being rather poor as well. Further

time is needed to flush these out of the ring buffer, which happens after 32 more measurements. This is expected to take somewhere around 80 seconds longer, and matches what is observed in figure 8.

Up to this point in the startup process, there is no real difference in startup speed between the two types of network conditions tested. This changes now as the clock quality estimation starts to be able to learn how good the clock crystal is. For the good network conditions, this clock quality estimation reaches a stable point at around the 10 minute mark. However, the larger network errors significantly slow the feedback available to this estimator in the bad network case, resulting in significantly slower convergence.

VI. Conclusions and further work

From the previous section we can see that both the measurement error and clock stability estimators function properly. They converge reasonably quickly towards estimates of the actual amount of noise, and stay stably at those estimates afterwards. The resulting clock servo then also manages to steer the clock to a degree that matches expectations given the network conditions and the resolution of the network card clock involved.

The resulting servo has no further parameters that depend on physical properties of the equipment in use. This leads us to believe it will likely perform well on a wide variety of underlying hardware with its default parameters. This property makes it particularly well suited for a general purpose implementation of PTP, as the user of the implementation needs no specialized knowledge to setup the software, whilst still getting a servo that is operating close to optimally for their hardware.

A limitation in our testing is that practical constraints only allowed us to test with a single type of oscillator. It would be interesting to repeat the measurement portion of this work on more hardware setups to see if any interesting variations surface. The author has been able to do this to a degree as part of the development work on Statime, but not in a fully controlled environment.

Of note is that the servo does require some amount of time at startup to get to a properly synchronized clock. Based on the startup behaviour observed, this seems to be dominated in large part by the estimate of the measurement error. It would be interesting whether there can be found alternative methods for the early estimation of the measurement error that are less conservative, as that could significantly speed up the startup process.

A second direction of improvement could be the estimation of the oscillator stability. The current method results in an estimate with a rather coarse resolution. It would be interesting to see if there are approaches that can provide a more direct estimate of the oscillator stability, rather than an indication of whether the current estimate is too high or too low. Such a direct estimate could both conceivably converge a bit faster as well as provide a more accurate estimate, further improving servo performance.

On the theoretical front, it would be interesting to consider more elaborate noise models. The model currently used does not include flicker noise components, and we do not dynamically infer a magnitude for the phase random walk noise. Including and dynamically estimating these noise components could further improve the quality of estimates.

VII. Acknowledgements

The author would like to thank Dr. B. Stienen, M. Schoolderman, M. Peeters, and R. Nijveld for discussions that made this work possible. Thanks also to the Sovereign Tech Fund for the funding which made Statime and part of the research for this paper possible.

References

[1] O. Obleukhov and A. Byagowi. How precision time protocol is being deployed at Meta. [Online]. Available: https://engineering.fb.com/2022/11/21/production-engineering/precision-time-protocol-at-meta/

[2] "IEEE standard for a precision clock synchronization protocol for networked measurement and control systems," *IEEE Std 1588-2019 (Revision of IEEE Std 1588-2008)*, pp. 1–499, 2020.

[3] R. Exel and F. Ring, "Improved clock synchronization accuracy through optimized servo parametrization," in *2013 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication (ISPCS) Proceedings*, 2013, pp. 65–70.

[4] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Journal of Basic Engineering*, vol. 82, no. 1, pp. 35–45, 03 1960. [Online]. Available: https://doi.org/10.1115/1.3662552

[5] C. A. Greenhall, "Forming stable timescales from the Jones–Tryon Kalman filter," *Metrologia*, vol. 40, no. 3, p. S335, jun 2003. [Online]. Available: https://dx.doi.org/10.1088/0026-1394/40/3/313

[6] A. Bletsas, "Evaluation of Kalman filtering for network time keeping," *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 52, no. 9, pp. 1452–1460, 2005.

[7] B. R. Hamilton, X. Ma, Q. Zhao, and J. Xu, "ACES: adaptive clock estimation and synchronization using Kalman filtering," in *Proceedings of the 14th ACM International Conference on Mobile Computing and Networking*, ser. MobiCom '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 152–162. [Online]. Available: https://doi.org/10.1145/1409944.1409963

[8] G. Giorgi, "An event-based Kalman filter for clock synchronization," *IEEE Transactions on Instrumentation and Measurement*, vol. 64, no. 2, pp. 449–457, 2015.

[9] D. Allan, "Statistics of atomic frequency standards," *Proceedings of the IEEE*, vol. 54, no. 2, pp. 221–230, 1966.

[10] D. Howe, D. Allan, and J. Barnes, "Properties of signal sources and measurement methods," in *Thirty Fifth Annual Frequency Control Symposium*, 1981, pp. 669–716.

[11] "IEEE standard definitions of physical quantities for fundamental frequency and time metrology-random instabilities," *IEEE Std 1139-1999*, pp. 1–40, 1999.