Clock synchronization problem can be seen as a model estimation problem. E.g. one can model a relation between local clock C_t

which is to be synchronized with a reference clock R_t

as R_t = offset + C_t + e_t

, where offset

is clock difference to be estimated and e_t

absorbs left-over errors.

Alternatively, one can pose the clock synchronization as a prediction of reference clock readings, based on an available time source. After the model is estimated (i.e. value of the offset

), based on sample readings, one can approximating the reference time, using the available clock (clocks) and the model E(R_t) = offset + C_t

.

In a robust estimation setting, when (a significant portion of) e_t

may be arbitrarily large, one can use median or other robust method, to estimate the offset

. When C_t

and/or R_t

are interval data, one can use Marzullo's algorithm or Brooks-Iyengar algorithm to obtain an interval estimate of the offset

.

The model above is extremely simple, basically, it has no (estimated factors, though prediction depends on a local clock, which rate is fixed and assumed to be known (for simplicity, we assume the rate is 1, if it is different but known, one can adjust C_t

definition appropriately).

One can re-arrange terms to make it more evident R_t-C_t = offset + e_t

.

In practice, more complex models are often required, as clocks are drifting, i.e. clock rate varies over time, but the variation can be predicted or explained by other factors, leading to more complex models.

A two simple extensions are:

- piece-wise-constant model, which corresponds to periodic re-estimation of the above constant model
- model clock rate as an estimated parameter, i.e. a simple linear model R_t = a + b*C_t + e_t

Both can be combined, resulting in a periodically re-estimated simple linear model (or a piece-wise simple linear model).

Other factors can be accounted for too, e.g. temperature and oscillator aging. However, robust model estimation requires too many computational efforts for multi-factor model, so such models are perhaps excessive in our setting.

If R_t

and C_t

are point estimates, then well-known robust estimators can be used, to estimate the simple linear regression model, e.g. Theil-Sen estimator, repeated median regression, RANSAC, LAD regression, Huber-regression, etc.

But what if the sampled data are available with interval estimates (e.g. seeSensor Fusion for BFT clock sync) ? We propose to adapt existing simple linear regression estimators, which are build around median estimator, where the median estimator is replaced with an analog which is able to work with intervals, e.g. the before-mentioned Marzullo's algorithm or Brooks-Iyenga algorithm, though other approaches are possible.

Why better clock models are useful

Before going deeper into estimation of linear models using interval data, let's discuss reasons to use more complex models

in block chain clock synchronization context.

A local clock is an important ingredient of a clock sync protocol, as an attacker cannot control it. The other aspect is that it allows keeping time in-between synchronization with a reference clock.

Any local clock is inevitably drifting relative to a reference clock and the speed of the drift is both important to detect attacks and make periods between clock re-synchronization longer (thus reducing average clock synchronization costs).

Adversary power limitation/attack detection

Let's assume an adversary has no control over local clock of an honest/rational node. Assume also, that an adversary can eclipse an honest node, however, for a limited period of time (e.g. sooner or later the node administrator will detect the eclipse attack and will be able to step in).

So, if an adversary tries to affect node's time estimates by supplying big errors, then the node can compare it against its local clocks - if they are too big or too low, then it's an obviously erroneous time reading and can be rejected.

So, a more realistic strategy would be to ty to adjust node clocks by smaller amounts - which are indistinguishable from nominal clock drift. As we assume that a normal clock can drift 100ppm, this is the rate that an adversary should aim at.

But that also means that if a node can estimate its clock better, then an adversary power becomes limited, as 100ppm drift can be detected as an error.

Time between re-synchronizations

Typical RTC drift is 100ppm - about 8 seconds per day. Due to regular re-synchronization it doesn't seem to be a problem. However, if we take two local clocks (of two protocol participant nodes), their relative drift can reach 200ppm, assuming each individual clock is 100ppm.

Assuming that one wants to keep clocks of two participants within 500ms, then initially perfectly synchronized clock can diverge in about 2500 secs, in the worst case. That means participants should re-synchronize clocks during that period or even more often, since there is non-zero residual disparity.

So, if one can estimate the individual clock drift better, then synchronization period can be made longer. Which can reduce communication costs - or made it easier to piggyback existing message flow. Estimating relative drift of clocks of two participants can additionally increase accuracy/precision.

Better clock model by calibrating clock rate

THe aforementioned 100ppm is a typical nominal drift around a nominal rate (i.e. 1 second per second), i.e. some prior information. Mathematically, one can formulate that local clock rate is bounded: $1 - \delta \le {L(t_1) - L(t_2) \over t_1 - t_2} \le 1 + \delta$

, where t

is unobservable 'true' time, L(.)

is local clock reading and δ

is admissible rate variation (e.g. 100ppm).

In practice, an individual clock can be significantly more stable since its actual rate differ from nominal. For example, suppose a clock has rate 0.99991 ± 0.00001

, so the clock instability is 100 ppm, measured relative to a nominal rate of 1. However, relative to the 0.99991 rate, its instability is (almost) 10ppm.

So, if one estimates the actual clock rate then one can correct the clock and obtain clock readings with much better stability relative to the nominal rate of 1, given that measurement/estimation errors are low enough.

Communication and computation costs

In theory, clock sync protocol participants can calibrate/characterize their clocks (i.e. measure actual clock drift relative a set of clocks and estimate clock rate from the data). They can communicate measured rates to participant.

However, in BFT context such communication can be a problem, especially when amount of participants is huge.

Therefore, it may be easier for each node to calibrate perceived clock rates of other participants (along with calibrating own local clock). Again, as the number of participants is huge, that can be computationally expensive.

Clock estimation and synchronization interactions

So, a participant can in theory estimate clock models for:

- its local clock
- its reference clocks (e.g. NTP servers)
- “perceived” clocks of other participants (as they may communicate not their local clock readings but corrected ones)

Actually, a reference clock when it functions correctly is synchronized to extremely stable time source (e.g. an atomic clock), so their model need to be typically estimated to detect errors (e.g. attacks).

The situation is a bit more complex, as the three type of models should be estimated relative to each other, since we assume BFT context, so arbitrary faults are possible (and there is no single ‘true’ clock). That means model estimation is intertwined with clock synchronization and filtering of erroneous signals.

A very simple approach would be to estimate e.g. local clock model relative to a reference clock readings, when the last clock is correct (i.e. there is no attack, normal network conditions, etc). As reference clocks are expected to be highly accurate and stable, then a very good model of local clock can be obtained.

While the simple approach is useful to illustrate the intended usage scenario, there are problems with such approach in BFT settings:

- clock should be re-estimated periodically (as there factors like temperature and oscillator aging)
- in BFT context one cannot trust small set of reference clocks

A better but still relatively simple approach is to employ a BFT clock sync protocol, which works with default assumption about local clock drift (e.g. 100ppm relative to nominal rate of 1). Under appropriate assumptions, one can obtain a stream of reliable time estimates, which is a fused combination of the tree kinds of time sources (e.g. see here).

Then a single clock model can be estimated against the fused time source. Then better local clock models and models of perceived clocks of other participants can be used to shrink input estimates, as uncertainty due to clock drift is added up to input estimates. So, more accurate clock models lead to more accurate and precise final estimates (at least in the worst case, as when errors are uncorrelated they tend to cancel each other and net effect can be quite low).

This results in a form of recursive estimation, when a robust clock sync protocol allows estimating better clock models, which can improve accuracy and precision of the clock sync protocol, which in turn can improve clock models.

A model of reference clock (relative to the fused time source) can be estimated too and used for fault detection, i.e. if an observed reference clock rate differs too much from the nominal or advertised rate, then the reference clock can be excluded from the set of reference time sources. The details of the fault detection can be left to particular implementation (e.g. a node can drop bad behaving NTP nodes from its pool and/or look for fresh ones).

Robust estimation of linear model using interval data

Interval data are quite useful in robust setting, especially, when interval widths can vary (i.e. different data points can have different interval widths). For example, clock readings from different reference clocks can arrive at different point of times, that means adjustments due to clock drift have different magnitude (i.e. proportional to time passed from the moment of a message arrival). If all widths of intervals should be the same, one should choose the maximum interval width, which can lower accuracy/precision. Another useful property is that uninformative data is naturally modeled by very wide interval width (e.g. (-\inf, \inf)

).

For this reason, it is desirable to develop a robust linear model estimation method to work with interval data.

One approach can be to take a robust linear method estimation built around a median estimator (e.g. Theil-Sen or repeated median) and replace median with Marzullo’s or Brooks-Iyengar algorithms.

Let’s decompose the methods first. One can estimate a simple linear model (y_t = a + b*x_t + e_T

) using a two-step approach:

- estimate slope b

first, using data point differences (y_{t2}-y_{t1},x_{t2}-x_{t1})

, e.g. one can use quotient y_{t2}-y_{t1} \over x_{t2}-x_{t1}

as a slope estimate for each pair. The set of pair-wise quotients can be averaged, e.g. using a median estimator.

- create a new factor using the slope estimate b*x_t

, and estimate a

based on that, e.g. using differences y_t-b*x_t

, again with a robust estimator, like a median.

NB

OLS can be decomposed in a similar way, with the difference that slope is estimated using weighted average. A similar approach can be used for robust estimation too.

Summarizing, it is a reduction of a simple regression (one factor model) estimation problem to a couple of constant model (zero factor models) estimation problems - one for slope and another one for offset.

Both Theil-Sen and repeated median estimators fall into the category, using median estimator as a tool to estimate constant models. The main difference between them is that Theil-Sen constructs such quotients for all pairs (e.g. n(n-1) \over 2

), whereas repeated median estimates the slope in two steps:

- for each data point, calculate n-1

quotients using this data point and each other data point, then apply median to them to obtain a slope estimate at this data point

- on the next step, the estimates obtained on the previous step are averaged using median again, obtaining final slope estimate for the whole data set

Repeated median is a more robust estimator than Theil-Sen, itsbreakdown point is 50% vs ~29% for Theil-Sen estimator (link), so it may be preferred.

In order to adapt them to work with interval data, one should replace median with Marzullo's or v-Iyenga algorithm (or some other similar method) and adapt the slope estimate y_2-y_1 \over x_2-x_1

to interval data setting.

Single point slope estimator

Single point slope estimators (e.g. y_2-y_1 \over x_2-x_1

) should be adapted for interval data.

If x

's are point estimates, then there is no problem: one can output [min {y_2-y_1 \over x_2-x_1}, max {y_2-y_1 \over x_2-x_1}]

as an interval slope estimate.

This is the case for our approach to clock synchronization as we do not use interval estimates for local clock readings (they are sampled rare enough, so local clock reading uncertainty is negligible or can be accounted for by other means).

In general, however, x

's can be intervals too. For example, one can reduce a problem of estimating a multiple factor linear model to a series of simple linear models, using orthogonalization process. E.g. one can choose one factor and regress output variable and other factors on it. Then calculate residuals and explain output variable residual by factor residuals. The problem is that the factor residuals will become interval estimates in the setting.

So, even if original factors are all point-wise estimates (zero-width intervals) in a multi-factor model, one may need to work with interval factors.

If x

's intervals do not intersect, then it's actually not a problem, since one can calculate maximum and minimum slope value for each pair of data points. It's more tricky when x

's intersect - obviously, it means slope can have any value - i.e. (-\inf, \inf)

. However, it's not a problem to adapt Marzullo's algorithm to work with them - basically, such point is uninformative, and one

can simply replace it with the worst bound values, (e.g. replace -\inf

with min(x_i.lower)

and \inf

with max(x_i.upper)

).

We have not yet performed rigorous analysis of the approach and don't know limits of its applicability, but in the case of point-wise x

's data (clock sync protocol) it worked fine.

Using prior information

We assume that each correct local clock drift is limited, e.g. 100ppm relative to a nominal rate of 1 second per second. In the context of clock model estimation this is a prior knowledge (before we experienced how the local clock behave relative to a reference clock).

It can be used during model estimation, like regularization is used in e.g. OLS estimation.

Basically, the prior bounds possible clock drift and when we calculate a slope estimate for a couple of readings, e.g. max/min of y_2-y_1 \over x_2-x_1

, we can constrain it using the prior information. E.g. if both clock can drift 100ppm, then interval width (relative to time elapsed) cannot be greater 200ppm (though some additional correction may be necessary depending on assumptions), under assumption that both clocks are correct. As we assume the amount of incorrect clocks is limited, the possible violations of the assumption are accounted for by Marzullo's or Brooks-Iyengar algorithm (e.g. see for details here).

Performance considerations

Both Theil-Sen and repeated median estimators use all pairs combinations to estimate linear model, so, their brute force calculation approach requires O(n^2)

time. It can be improved using sophisticated approaches to O(n *log(n))

and O(n*log^2(n))

(e.g. 1, 2).

Actually, in clock sync setting, online estimation is required, so an update can be done with O(n)

steps (at least, for the repeated median regression).

It's acceptable for a single clock model or several models per node, since size of history can be set to a relatively low value (e.g. tens or a hundred).

However, it can be prohibitive if models of participants clocks are estimated too, as we expect a huge amount of participants (tens of thousands).

One aspect that can help to speed up model estimation is that clock fusion effectively filters out erroneous clock readings. So, a faster but less accurate estimation is admissible. For example, one can limit set of pairs, e.g for each data point consider a constant small amount of other data points to construct a slope estimate at the data point (e.g. 1-5). In the case of Theil-Sen, that can be incomplete Theil-Sen estimator.

Sample implementation

The approach is implemented in Kotlin - incomplete Theil-Sen, Theil-Sen and repeated median estimator for simple linear models, adapted for interval data (y

's only).