This is a follow up of

Time as a Public Service in Byzantine context and

Time attacks and security models.

Many BFT and Proof-of-Stake protocols rely on a lock-step execution, which in practice means that node's clocks should be synchronized within certain bounds. On-board clocks are typically not stable enough, which means periodic re-synchronization is required.

But if clock synchronization is implemented via NTP protocol, which is the cheapest and easiest option (and looks like the most viable in context of open/permission-less systems), then there arises a risk of NTP-level attacks.

Thus, any practical BFT protocol implementation, which relies on a lock-step execution should be augmented with BF-tolerant clock synchronization (sub)protocol.

While it's not difficult to design a protocol to synchronize clocks in non-Byzantine context, dealing with Byzantine faults can be challenging. This is especially true in the case of beacon chain protocol, where there are lots of active entities: both validators (300K validators expected, up to 4M max) and nodes (up to 10K nodes expected).

Relying on traditional approaches might not be enough to attain reasonable accuracy and precision while being resilient to multiple Byzantine faults. One opportunity is to use Public Time Services. Another one is to exploit inherent randomness (e.g. clock drift, network delays), which is difficult or impossible to control by an attacker.

Additionally, for a practical system it may be highly desired - or even required - to have its internal time synchronized with a World Time Standard (e.g. UTC, typically).

In BFT context, arbitrary faults are possible, which means worst case analysis (under various security models, restricting adversary power) is required.

In general, introducing clock synchronization protocol can result in heavier load on existing resources. It's highly desirable that additional costs incurred by clock sync protocol are minimal.

Overall, designing a practical clock synchronization protocol for BFT solutions can be a challenging problem. To address this, we propose Sensor Fusion Framework, which unifies several similar approaches, so that one can re-use and mix existing approaches (e.g. algorithms, proofs of their properties, etc). These underlying approaches are:

- Inexact / Approximate Byzantine Agreement protocols
- Time Keeping protocols (e.g. NTP)
- Sensor Fusion (clocks can be seen as sensors)
- Data Science (e.g. robust statistics)

## Sensor Fusion

Clock Synchronization problem can be seen as Sensor Fusion, which is similar to Approximate/In-exact BA. Sensor Fusion approach gives a broader view, which has several benefits.

First, one can use local clocks as well as reference clocks (that should be) synchronized with time standard. Both types of clocks can be seen as sensors. Besides that, any entity keeping or providing time (time server, validating node, etc) can be seen as a sensor and, thus, viewed in uniform framework (network of sensors of different kinds).

Second, it leads to a very simple fault model, where the only fault is a clock reading is beyond admissible bounds. That includes two-faced clocks, where different clock offsets can be introduced when reading the same clock via different links (by different processes).

Third, the most important is that there are sensor fusion algorithms with known accuracy and precision bounds. This is very important for assessing clock synchronization algorithms in BFT context.

An additional, fourth benefit is that there are also sensor fusion algorithms which works with interval data (e.g.Marzullo's algorithm, Brooks-Iyengar algorithm

). Since, intervals are very natural to describe clock readings and different clocks (read via different paths) can have different width of intervals, such interval fusion algorithms can lead to better accuracy/precision in a straightforward way (no need to take widest interval to calculate precision/accuracy bounds common for all clocks).

## In-exact/Approximate Byzantine Agreement

Inexact / Approximate BA protocols are relatively simple:

- distributed processes send their estimates to each other
- each process applies robust aggregation function to the estimates
- the procedure can be repeated for better accuracy/precision

There can be variations, e.g. Crusader Agreement

to filter out two-faced faults, however, they require much more data to exchange, so we will ignore them mostly.

Convergence clock sync protocol can be build on top of (iterative) Approximate/Inexact agreement. An important property is that very simple broadcast protocol is enough to exchange clock readings, because any omission/delta can be modeled by introducing an offset to the clock difference transferred.

## Employing Sensor Fusion Framework for clock sync

Each node (distributed process) has two types of clocks:

- a local clock (or several of them)
- remote reference clocks (NTP servers, GNSS Receivers, Radio Wave clocks, etc)

Clocks can be faulty, but their faults are different. We assume, local clocks (of uncorrupted nodes) cannot be controlled by an attacker, however, they are drifting relative to other clocks (including remote reference clocks and local clocks of other nodes, so they are incorrect in a long term. Remote reference clocks are synchronized to World Time Standard (e.g. UTC), however, an attacker can corrupt part of them or corrupt links they are attached with.

As we also assume that adversary power is limited (e.g. minority of clocks/nodes are corrupted), the goal is to exploit the clock/fault diversity, i.e. gather clock readings from protocol participants and fuse them into a single reliable source of time (per each correct/honest node).

In general, given suitable clock exchange protocol, each node has four time sources:

- own local clock
- own reference clocks (can be multiple)
- local clock readings of other nodes
- reference clock readings of other nodes

Then such node can implement some sensor fusion strategy/algorithm to obtain a time estimate. So, overall problem is split in two sub-problems:

- clock exchange protocol
- time sources aggregation strategy

In general, final time estimate can be reached via an iterative process (e.g. nodes exchange their intermediate estimates).

Additionally, in BFT context, we need to analyze worst case behavior of the resulting protocol. Two main aspects:

- worst case precision, i.e. upper bound on final time estimates disparity between honest/correct nodes
- worst case accuracy, i.e. upper bound on final time estimates disparity relative to true value (i.e. World Time Standard, in our case)

Note, that inherent randomness - local clock drift or network delays - can be exploited to certain degree. Basically, if an attacker provides nodes with clock readings which are way off true value, then it’s easy to spot and filter out. Thus, the attacker should provide precisely calculated values, to maximize final estimates disparity. But precise information on values gathered by honest/correct nodes is required to do that - e.g. knowing relative local clock drift, exact message delays, etc - which is difficult (or impossible) to measure.

Therefore, a security model can reasonably made certain assumptions on uncertainty of information (about correct/honest nodes) available to an adversary.

## SF Model

Let's formulate Sensor Fusion model (as it's seen as a part of Sensor Fusion Framework). The model itself is rather simple: there are sensors and processing elements (PE), connected by links.

## Sensors

An important aspect is that sensor outputs an interval, containing true value (in general, it can be a probability distribution). However, sensor readings can be faulty - i.e. an interval doesn't contain true value. A fault can be introduced by a sensor or by a link. The last case models two-faced sensors, which outputs different values when read by different processes. In the case of interval outputs, it's not always clear which values are different.

We distinguish two approaches:

- accuracy-based - two intervals are similar if they both contain or do not contain true value
- precision-based - two intervals are similar if their intersection is non empty

## PEs and connection topology

A PE is connected to an array of sensors, reads their output, processes them and produces some output in result. We assume, is connected to

relationship is many-to-many, e.g. same sensor can be read by several PEs.

As PE may aggregate or relay sensor readings, it can be seen as a soft or virtual sensor and thus, other PEs can be connected to it.

That results in a DAG, where terminal nodes are "physical" sensors.

In general, interactive/recursive sensor fusion can be performed, i.e. intermediate estimates can be fed back, which may need a cyclic graph to express, but we ignore this currently. We assume that recursive/interactive estimation is performed by keeping a history of results received on previous rounds of communications, e.g. PEs may have state.

Typically, a PE averages readings of its sensors. However, in general, it can output any value (set of values), e.g. output multiple averaging results, partial results, relay sensor readings, etc.

PE can aggregate "spatial" readings of an array of sensors or "temporal" readings (i.e. history of a sensor or sensor readings).

## Replicated PEs

We assume there may optionally be replicated set of PEs in the sense, that they are connected to similar arrays of sensors (the arrays can be overlapping but can be non-overlapping as well) and computing similar outputs, which we intend to compare between each other.

E.g. in case of clock synchronization protocol, each node may act as a soft sensor (providing some intermediate estimate) and as a PE providing final time estimate (to be used by its beacon chain protocol instance). To judge the overall protocol, we need to analyze worst case bounds on precision and accuracy of these final time estimates. So, both final PEs as well as intermediate PEs can be seen as examples of replicated PEs.

## Fault kinds

The only fault kind is when output interval doesn't contain true value. However, other faults can be modeled by special kinds of intervals, e.g. an empty interval, a very wide interval, an interval beyond reasonable range of values, etc. As noted above, a fault may happen in a sensor or in a link, e.g. a valid sensor can provide different values to different PEs.

However, a noise introduced via a link doesn't necessarily result in a visible fault, since it can be low or compensated by a sensor fault, so that the end result is in admissible bounds.

## Aggregate Functions

In BFT context, where an adversary may optimize values of corrupted sensor readings (given certain constraints), one has to use robust aggregation approach. There is a number of such approaches:

- order statistics, rank statistics, L-estimator, M-estimators
- robust regression methods (Huber regression, LAD regression,Theil-Sen estimator, Repeated median regression, RANSAC, etc)

- Marzullo’s algorithm, Brooks-Iyengar algorithm
- aggregating functions used in Inexact / Approximate

BA approaches (FCA, CCA, etc)

Some of the approaches (Marzullo’s algorithm, Brooks-Iyengar algorithm) work with interval data and can output an interval. One can adjust robust regression to employ the algorithms instead of a median to apply to a simple linear regression setting.

## Examples

As the main reason to introduce SF framework is to facilitate re-use of approaches, algorithms - and proofs of their properties - from related areas, let’s explore briefly notable examples and how they can be expressed in SF framework.

### Disciplined XO

An ubiquitous timekeeping approach - disciplined oscillator - can be seen as an example of SF.

There are two sensors: a local clock (oscillator + counter) and a reference clock.

It’s assumed that local clock has good short term stability but loses accuracy in a long term. The reference clock is stable in long term, but it can be difficult or impossible to access the clock randomly or too often.

The solution is to read local clock, but correct its drift by comparing it with the reference clock periodically.

The simplest solution is to adjust clock offset only. A more accurate approach is to have a more complex (local) clock drift model and estimate its parameters by comparing with the reference clock. In addition to offset the model parameters can include clock rate (first derivative), crystal aging parameters (second, third, etc derivatives), temperature. Accounting for temperature changes requires a temperature sensor.

In the context, sensor fusion can be seen as a machine learning/statistics problem: learn predictive model of reference clock, given history of local sensor readings.

UTC time standard keeping is organized in a very similar way - there is a bunch of very stable clocks and frequency standards, the clock readings are then fused into common time, which is more stable then any component.

### Robust clock calibration

Machine learning often assumes independent errors. Even if correlation is assumed it’s typically not appropriate approach in BFT context, where an adversary can choose any schedule within limitations of a security model.

Sensor fusion based on simple averaging is not robust enough to tolerate a powerful adversary.

Robust statistics should be employed thus. There are many robust estimators, the main problem being they often require lots of calculations.

However, in simple cases that can be O(N) or O(n log n), and even O(N^2) can be acceptable if N is relatively small.

Let’s explore two simple cases: constant model (clock offset only) and simple linear model (clock offset + clock rate) robust estimation.

#### Offset estimation

Let’s assume local clock drift is bounded, i.e. clock rate is $1 \pm \rho$

. As only clock offset is estimated, the model is $y_t = a + x_t + e_t$

, where $y_t$

is the reference clock samples, $x_t$

is corresponding local clock readings, $a$

is unknown (to be estimated) clock offset and $e_t$

is errors. In BFT case, $e_t$

is not bounded, however, we assume that it’s small in majority of cases, however, an adversary can corrupt arbitrarily minority of samples (we can split $e_t$

in two parts, for example).

An obvious idea would be to try median, i.e. $\hat a = median(y_t-x_t)$

. As we assume that only minority of samples is unboundedly corrupted, then $\hat a$

is either equal to a “correct” $y_t-x_t$

or between to “correct” $y_t-x_t$

, where by “correct” we assume a sample where $e_t$

is small (within admissible bounds, e.g. not corrupted by an adversary).

If the samples are intervals (containing true value), then Marzullo or Brooks-Iyengar can be used (can output an interval as a result too).

However, precision of the estimate may be unsatisfactory in the replicated PE setting. There are many distributed process estimating clock offsets and two-face faults are possible, then adversary can send too big readings (greater then any correct sample) to one part of PEs and too small readings (smaller then any correct sample) to the rest of PEs. All PEs will output values that are in the range of correct samples. However, the adversary forces first group of PEs to choose highest values while second group - lowest values. In worst cases that means, the range of PEs outputs won’t shrink (e.g. when there are multiple lowest values and multiple lightest values), that means:

- convergence is not always possible (when needed)
- worst case precision is not the best possible

There are better approaches possible from worst case perspective (additional or stricter assumptions may be required):

- Inexact Agreement
- Approximate Agreement

**Offset + skew estimation**

One often assumes local clock drift is bounded (e.g. non-faulty rate is $1 \pm \rho$

), however, bounds can be chosen loose to accommodate for real oscillator rate variations relative to a nominal rate, e.g. $100ppm$

relative to nominal rate of 32,768 KHz. It can be the case that local clock’s oscillator is much more stable, meandering around some unknown rate, which differs from the nominal.

The rate instability bound is important since the less this bound the longer periods between clock re-synchronizations are possible, which means less communication overheads (and longer paths between nodes in p2p graph are acceptable).

Thus, a two-factor model of a local clock can be employed, e.g. $y_t = a + b*x_t + e_t$

, where $y_t$

and $x_t$

are reference and local clock samples, respectively, $e_t$

is possibly corrupted noise (i.e. small noise plus minority of arbitrarily corrupted samples) and $a$

and $b$

are model parameters to be estimated: clock offset and clock rate respectively.

A robust simple linear regression estimator should be used to estimate clock rate, e.g. LAD regression, Huber regression, Theil-Sen estimator, Repeated Median Estimator, etc.

In the case of interval data, one can adapt Theil-Sen or Repeated Median Estimator to useMarzullo’s or Brooks-Iyengar

algorithms instead of median. Trimming can be also used to fuse prior knowledge (e.g. nominal rate bounds).

## Clock Sync protocols

Many clock synchronization protocols (including BFT ones, e.g.1, 2, 3, 4) in academic literature can be seen as instantiations of Sensor Fusion Framework. Distributed nodes have local clocks attached, then they use some form of Broadcast to exchange clock readings adn employ some robust averaging function. The process can be iterated, optionally.

So, each node can be seen as a combination of a physical sensor - clock and a PE. Each process can read local clock of each process (including own), in SFF parlance, each PE (one facet of each distributed process) can read values from each sensor (another facet of each distributed process.

In a more complex approaches (e.g. iterated estimation), local clocks are not exposed directly, instead each node acts as two PEs:

- a soft sensor providing current estimates to other nodes (initialized with a local sensor reading)
- final PE, aggregating soft sensor estimates (which are also used to update corresponding soft sensor)

### NTP protocol

NTP protocol is organized as a hierarchical layered system of time sources. Top-level (stratum 0) time sources, also known as reference clocks in NTP, constitutes the root(s) of the hierarchy. Each lower-level (higher stratum) time source is connected to higher-level (lower-stratum) time sources and/or to its peers of the same level/stratum. Thus, each non-zero stratum time source has a set of time sources, which it uses to disciplined its local clock.

From SFF perspective, stratum 0 time sources act as physical sensors, while each non-zero stratum time source is a combination of a physical sensor (local clock) and a soft sensor/PE fusing local clock readings and history of readings of time sources it’s connected to.

NTP uses a variation of Marzullo’s algorithm to filter out false-ticker

time sources and use some statistical approach to robustly discipline its local clock. The disciplined local clock readings are either broadcasted or sent by request to its peers or clients.

# Mapping Clock Sync problem on SF

System model where clock synchronization is to be implemented is described in more detailshere. In two words: traditional distributed processes (nodes) connected by channels are augmented by special kind of processes - Public Services, and special kind of links. The special kinds of nodes allows to specify structured/correlated fault models.

In the case of Clock Sync problem, there is one kind of Public Service, which a Reference Clock (or Time Server). It can be NTP server, GNSS Receiver, Radio Wave Clock, etc.

Each node has also a local clock (or local clocks), which can be read by the node only (and thus it cannot constitute a two-face clock and its faults are not correlated with other process faults).

## Clock reading

If a clock is remote then reading it requires message passing. There can be several patterns.

### Simple time transfer

A sender records sending timestamp (reads its clock) just before sending a message and includes the timestamp in the message. Upon receiving the message, receiver records receiving timestamp (reads its clock).

NB

In some cases, a sender can obtain sending timestamp after sending a message, then an additional message may be required. In our setting, we assume it’s fine to read a clock before sending a message to record a sending timestamp. Any delay between to events is either insignificant or can be accounted for during estimation process.

The simple interaction allows the message receiver to estimate sender-receiver clock offset, given that message delay bounds are known.

### Round trip

A round trip pattern can be used to estimate message delay. I.e. an initiator sends a request and records sending timestamp. A responder records receiving timestamp, records response sending timestamp and sends the values back to the initiator. The initiator records response receiving time. The four time values can be used to estimate clock offset as well as message delay. It can be seen as a combination of two opposite simple time transfers.

### Triple time transfer (bidirectional)

In the round trip, only the initiator has enough data to estimate both clock offset and message delay. But it can send a third message to provide the responder with the information, so that both parties can estimate message delay.

### Repeated time transfers

Network delay can vary, so it make sense to repeat clock reading procedure several times, to allow for shorter delays to occur, which will improve measurement accuracy. One then can use min(rec_t - send_t)

as an estimate.

However, if the repeated time transfer happens over a long period of time, then participant's local clocks can drift away significantly. E.g. given \pm 100ppm

clock drift bounds, over 1000 second period, clock disparity can grow by up to 200ms

. The clock drift can be accounted for with Marzullo's algorithm and adjusting interval bounds appropriately. E.g. if there are two clock offset interval estimates (including network delay bounds, sender and receiver clock reading inaccuracies, but ignoring relative clock drift) - [2.0s, 3.0s] and [1.5s,2.5s], then ignoring relative clock drift, one can conclude that [2.0s,2.5s] interval contains true clock offset. However, if second measurement has occurred 1000s later than the first one and maximum clock drift is 100ppm

, then first interval should be replaced with [1.8s, 3.2s], so intersection of two intervals results in a slightly wider [1.8s, 2.5s] estimate.

Faulty intervals (e.g. those that do not intersect with others) can also be excluded with Marzullo's algorithm.

### Implied timestamps

In many protocols based on lock-step execution participants should send their messages in predefined moments of time, so that implied time of message sending can be calculated from the message contents. While explicit timestamp recording promises better accuracy, implied timestamp can be simpler and reduce communication costs (that can be important if there are many messages which are aggregated, e.g. aggregated attestations using BLS aggregate signatures).

## Fault Model

A useful property of SFF model is that there is only one kind of fault (an interval doesn't contain true value). That means other faults should be mapped appropriately (if it's possible). It's introduced since it simplifies subsequent reasoning and estimate calculations. Needless to say, it can be relaxed or ignored.

There are different fault kind possible:

- node crash/stop
- corrupted/dishonest node
- message omission
- too long or too short network delay
- clock offset beyond admissible bounds (clock rate can be in bounds)
- clock drift beyond admissible bounds (e.g. beyond 1 \pm \rho

)

- corrupted message

We assume a message cannot be forged, i.e. an invalid message is simply rejected, so it's equivalent to the message omission (in general, a container can fail verification check, however, it's content can be valid, so, depending on implementation, valid sub-messages can be used).

Node crash/stop means messages are not sent when required, which can be modeled as message omission again.

Message omission can be seen as an infinite message delay or a message delivered too far in future, so it can be seen as a message delay fault.

As message delay bounds are used to estimate an interval to which sender-receiver clock offset should belong, if a message travelled longer or shorter than expected, it means that it's possible that true value does not belong to the estimated interval, which means the message delay fault can be mapped to the our primary fault kind (an interval doesn't contain true value).

Clock offset is another example of primary kind of fault.

Clock rate bounds are also used to estimate an interval, which should contain sender-receiver clock offset, so if real clock rate fall out of admissible bounds it also may result in the case, when resulting interval doesn't contain the true values, which is a primary kind of fault.

If an adversary corrupts a node or a reference clock or a link, it can:

- delay or omit messages
- garbled messages
- populate it with wrong data

All the cases can be mapped to the primary kind of fault: i.e. delayed, omitted or garbled messages are already analyzed. As we are using timestamps or implied timestamps of messages (and an adversary cannot forge signatures of honest/correct nodes), then the only fault that an adversary can induce is the primary kind of fault - e.g. an interval may not contain true value.

## Conclusion

We presented a Sensor Fusion Framework, which is to be a paradigm to design a Clock Synchronization (sub)protocol suitable for augmenting beacon chain protocol - or any other BFT protocol which is based on a lock step execution.

The design of a lightweight Clock Synchronization protocol itself is to be discussed in follow up posts.

## Links

### Time

NTP

NTP Poll

UTC

GPS disciplined oscillator

### Clock Synchronization

Synchronizing Clocks in the Presence of Faults

Synchronizing Time Servers

Inexact Agreement

Dynamic Fault-Tolerant Clock Synchronization

### Agreement

Inexact Agreement

Approximate Agreement

Crusader Agreement

### Interval/Sensor Fusion

Sensor Fusion

Marzullo's algorithm

Brooks-Iyengar algorithm

Intersection algorithm

### Robust Statistics

Robust statistics

M-estimator

L-estimator

Theil-Sen estimator

Repeated median regression

RANSAC

## Ethereum Research related to Time

Why Clock Sync matters

Time Requirements

Potential NTP pool attack

Sybil-like NTP-level attack (eth2.0-specs issue)

Proposal against correlated time-level attacks

Time as a Public Service in Byzantine context

Time attacks and security models

Network-adjusted timestamps