Consensus Timestamps

Obyte improvement proposal (under the rules of OIP1 https://github.com/byteball/OIPs/blob/master/bbip-0001.md)

OIP: 0002
Title: Consensus Timestamps
Author: Tony Churyumoff @tonych
Discussions-To: discord #tech
Comments-Summary: No comments yet
Comments-URI: https://github.com/byteball/OIPs/wiki/Comments:BBIP-0002
Status: Draft
Type: Standards Track
Created: 2019-02-22
License: BSD-2-Clause and CC0-1.0

Abstract

An extension of Obyte consensus protocol is proposed which allows the nodes to reach consensus about the current timestamp. The value of consensus timestamp can be used by smart contracts without relying on oracles.

Copyright

Motivation

Nearly all contracts depend on a timestamp. For example, a contract definition often says that the contract address can be spent by one of the parties after some date and time in the future. There is no protocol-level timestamp in Obyte, hence contracts have to rely on a timestamp oracle. While it might be OK for short-term and/or low-stakes contracts, an oracle is still a single point of failure which should be avoided, especially for such frequently used feature as timestamp.

Specification

1. Each unit will now include one additional field: timestamp. It is the number of seconds since the epoch (note that the current timestamp oracles post the timestamp as the number of milliseconds).

2. The unit hash will include the timestamp. To make it possible to easily distinguish between the new units (with timestamp) and old (timestampless) units, the version field of new units will be equal to ‘1.1’ (‘11.t’ for testnet) as opposed to ‘1.0’ (‘1.0t’ for testnet) for old units.

3. All units whose last ball MCI is after a specific upgrade MCI (which will be set to a value well in the future to give all nodes enough time to upgrade) must be version ‘1.1’, all previous units must be version ‘1.0’.

4. Smart contract language is extended with additional clause which looks like [“timestamp”, “>”, 1550768343] and evaluates to true if the timestamp set in last ball unit is > (or >=, <, <=, =, !=) than the specified timestamp.

5. Timestamp of every unit must be larger than or equal to timestamps of each parent.

6. When a node receives a unit whose timestamp is in the future (according to its local clock):

    a. if the timestamp is not too far in the future (e.g. up to 10 minutes, a parameter configurable per node), it should accept the unit and forward it to peers as usual but not include it as parent in its future units until the timestamp “ripens”;

    b. if the timestamp is too far in the future, the node should temp-reject it. Temp-rejecting means that the node doesn’t accept the unit but can accept it in the future when its timestamp meets the acceptance criteria. Normally, when an invalid unit is rejected, it is permanently rejected, which means that its hash is also added to a list of rejected units (bad_joints table in the current implementation) and future resubmissions of the same unit are quickly handled by consulting this list rather than performing more costly validation again. Temp-rejection doesn’t add the unit to such list.

Rationale

Obyte is one of a few (the only?) consensus protocol that doesn’t include a timestamp. PoW requires a timestamp for retargeting the block difficulty, most PoS algorithms have a block production schedule and heavily rely on accurate clock on all block-producing nodes. Therefore, in such systems, a timestamp is already baked-in at the base protocol level, and smart contracts get it for free. Obyte, on the opposite, simply doesn’t need a timestamp for consensus, hence it is not baked-in.

However, as indicated above in Motivation section, smart contracts would benefit from a reliable and trustworthy timestamp.

Hence, the key design constraint of this proposal is to allow the nodes to come to consensus about the current time while not disrupting the underlying functionality of the network.

The proposed specification is similar to the rules of block timestamps in many popular blockchains, which are known to produce reasonably accurate timestamps, with the exception of time-warp attacks.

It is expected that the proposed rules will produce reasonably accurate (within a few seconds) timestamps for the following reasons.

First, note that the timestamp cannot go back according to item 5 of the specification. However, this rule still allows it to lag behind the real time (or even stop advancing) or go into the future. We obviously cannot be sure that every node is following rules 1 and 6 even if its clock is absolutely accurate, because nobody but the node can read its clock.

Nodes on Obyte network operate under incentives similar to those of miners on blockchains. Like miners are interested to immediately start mining on top of a new block they received from other miners, nodes on Obyte are interested to select the newest units as parents in order to be the first who takes them as parents and maximize their chances of winning the headers commissions paid by the parents. Like a miner who found a new block is interested in pushing it to peers as fast as possible to win the race against other miners who might have found the block at about the same time, Obyte nodes are similarly interested in pushing their newly created units to peers as fast as possible to make sure that other nodes see the new unit and select it as parent rather than selecting its parents instead and competing with the author's unit for the same headers commissions. They also need to make sure that their new unit ends up being on the same MCI or MCI+1 as the parent's MCI, otherwise if it comes later, it loses the chances of winning the headers commissions.

In light of the above, there is no point posting units with future timestamps because the honest nodes would simply delay them while the MCI continues to advance forward, therefore the author will stand no chance of winning the header commissions from his parents.

If we assume that the timestamp is considerably lagging behind the real clock time and even a single honest node exists who posts a unit with a larger timestamp, this unit is going to be quickly taken as parent by other nodes in an attempt to grab its header commissions. But the child nodes will have to at least adopt the increased timestamp of this unit according to rule 5 above, therefore the lag decreases.

Nodes might be cautious about posting the real timestamp knowing that some nodes might have a lagging clock and will see the unit as coming from the future and therefore will delay its parenting. Hence, the posted timestamps might be on average slightly lagging behind the real time. But not lagging too much, because as soon as the distance from the real time becomes larger than a typical discrepancy in clock time among nodes, it becomes safe to decrease the distance.

To compensate for the likely systematic lagging of the posted timestamp behind the real time, I considered allowing acceptance of units that are slightly (by a few seconds) in the future. Some blockchains already allow blocks with slightly future timestamps. However, this relaxation of the rules would create difficult choices for subsequent units: should they lie

about their timestamp and report a future time when they include a future-dated unit as a parent (remember, the timestamp cannot go back) or just filter out the future-timestamped units? Or should we allow the timestamp to go back by small increments? And what is the right delta between the real timestamp and the clock time that should be allowed? My design decisions are guided by avoidance of arbitrary constants like this. And the purpose of achieving slightly more accurate timestamps didn't seem worth the additional complexity. Anyway, the accuracy better than the typical clock time discrepancy probably doesn't make sense (arguably, accuracy better than the average confirmation time doesn't make sense either), so I ditched the idea.

Rule 6a ensures that units that appear to be future-timestamped due to normal discrepancy of clock times are still eventually accepted without requiring any action from the author. If we temp-rejected them, such units would have been quickly forgotten by all nodes on the network except the author, and the author would have to bother about monitoring it and rebroadcasting as required. This would contradict the basic property of a normally functioning payment system that the author should be able to fire-and-forget and be confident that the transaction goes through. However, the units that are extremely far in the future (rule 6b) are not likely to be the result of an honest mistake and should be temp-rejected to conserve resources.

Backwards compatibility

It is a breaking change and all nodes will have to upgrade before the upgrade MCI as indicated in rule 3.