A Distributed Clock Synchronization Protocol for Proof of Stake Blockchains

Yuya Miki†, and Kazuyuki Shudo†,††
†*Tokyo Institute of Technology, Tokyo, Japan*
††*Kyoto University, Kyoto, Japan*

***Abstract*—In a Proof of Stake (PoS) blockchain, all nodes need to be able to recognize approximately synchronized clocks. A PoS blockchain relies on external clocks, and each node synchronizes its local clock according to an external clock using Network Time Protocol (NTP) or a similar protocol. However, this external dependence is undesirable because it reduces the autonomy, sustainability and trustless nature of the blockchain. This paper presents a clock synchronization protocol for slot-based PoS blockchains. The proposed protocol estimates the block propagation time to ensure that the synchronization is closer to that of a real-world clock. In addition, the proposed method considers the clock drift and clock adjustment frequency to achieve more accurate synchronization. Simulation experiments show that the proposed method can synchronize with a time that is closer to that of a real-world clock and with higher accuracy than existing methods.**

***Index Terms*—Blockchain, Clock synchronization, Block propagation time, Ethereum**

I. Introduction

A blockchain is a type of distributed ledger technology that records the transfer of asset rights. A transaction represents the transfer of asset rights, and blocks are used to store a number of transactions. The blocks are connected by hashed values, which prevents the falsification of past data. Blockchains allow values to be transferred in the absence of a trusted third-party. Bitcoin and Ethereum are two prominent examples of blockchains [1], [2].

In a Proof of Stake (PoS) blockchain, a mechanism is needed to ensure that each node in the network recognizes the same round. Because typical crystal real-time clocks (RTCs) are not very accurate, for each node to recognize a round based on its own local clock, a high-precision clock such as an atomic clock or adjustment with an external clock synchronization protocol is necessary. However, high-precision clocks are expensive, and imposing hardware requirements on nodes raises the barrier to participating in the protocol.

As a result, many PoS blockchains use external clock synchronization protocols, such as Network Time Protocol (NTP) [3], to synchronize the time with a local clock. For example, in Ethereum, the most popular client software, Go Ethereum (Geth) [4], imposes a local clock requirement on each node that joins the network. This requirement is based on the time obtained by NTP. However, several problems have been reported as a result of reliance on NTP, which may compromise the security of the system. In addition, reliance on external protocols reduces the viability of the blockchain protocol as an autonomous decentralized system.

This work was supported by JSPS KAKENHI Grant Number JP21H04872.

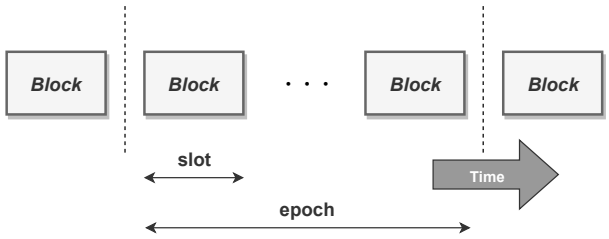

Fig. 1: Epoch and Slot in Ethereum 2.0.

Thus, in this paper, we propose a new clock synchronization protocol for slot-based PoS blockchains. The proposed method considers the effects of the block propagation time, clock drift, and clock adjustment frequency, which have not been fully considered in existing methods, to achieve synchronization closer to real-world clocks and with higher accuracy than existing methods. The proposed method can be implemented without imposing any additional communication requirements for clock synchronization and without placing a significant burden on existing protocols because this method considers only the received time of the blocks and the information in the block.

This remainder of this paper is organized as follows. In Section II, we provide an overview of existing methods for clock synchronization in PoS blockchains. Next, we describe the details of the proposed method in Section III, and evaluation experiments for the proposed method and their results are discussed in Sections IV and V. Finally, in Section VI, we present our conclusions and directions for future work.

II. Related Work

In general, PoS blockchains rely on external clock synchronization protocols. However, several studies have proposed clock synchronization protocols to reduce this dependence on external clock synchronization protocols. This section describes clock synchronization protocols that have been proposed for PoS blockchains. We first describe the existing methods for Ethereum and then discuss other methods.

A. Clock Synchronization Protocols in Ethereum

Before we describe the existing methods, we explain the concept of rounds in Ethereum 2.0, an upgraded version of

Proc. 5th IEEE Int'l Conf. on Blockchain
(IEEE Blockchain 2022), August 2022

Ethereum. In Ethereum 2.0, time is divided into epochs and slots, as shown in Figure 1. One slot is defined as 12 seconds, and each epoch is divided into 32 slots. A randomly selected validator generates and proposes a block in a given slot.

Buterin [5] proposed a method for synchronizing the local clock of each node using the received time of the most recent blocks received from neighbor nodes and the information in the blocks. The local clock of the node that generated the block is estimated according to the timestamp of the genesis block, the slot number stored in the block, and the slot length, and the offset between the estimated and received times of the block is calculated for all nodes. Then, the node uses the median value to adjust its own local clock. Simulation experiments have shown the effectiveness of this method. However, the effects of network latency and clock drift are not taken into account. In addition, the frequency of clock adjustment is either too high or not considered, and the freshness of the time information used to calculate the estimated time may be low.

Vlasov [6] proposed a method for mapping the clock synchronization problem to sensor fusion that uses a well-known sensor fusion framework to estimate the time and filter the interval data. However, because of the possibility of introducing additional traffic to the existing network when obtaining the time information and the assumption that a reference clock synchronized to standard time exists, this method is beyond the scope of comparison of this paper.

B. Other Clock Synchronization Protocols

Symbol [7] designs and uses its own protocols and is thus independent of external entities. This protocol estimates the local clock offset according to neighboring nodes and synchronizes the time using a simple NTP-based procedure. Bad data are filtered based on the response time of the corresponding node, and each offset is weighted according to the importance of the node that reported it to prevent Sybil attacks. In addition, the number of clock adjustments is recorded, and the age of the node is considered to strictly control the behavior of older nodes. However, unique messages are needed for clock synchronization, which is a challenge that may generate additional traffic if this method is implemented in existing protocols.

Alper [8] proposed Relative Time protocol, a method that uses the estimated offsets of the local clocks of the block producer and itself, which are calculated according to the received times of the valid, finalized blocks received during the relevant epoch. The local clock is corrected during each epoch by taking the median of the estimated offsets. Relative Time protocol can be implemented without imposing a significant burden on the existing consensus mechanism since this protocol uses only the information in the block and the received time of the block, similar to [5]. However, the effects of the propagation delay and clock drift are not fully considered, and the amount of time information used to calculate the modification value may be limited because only information from the relevant epoch is used.

BeaconBlocks [9] is a PoS protocol with a unique clock synchronization method. This protocol proposes mechanisms for determining the correct time at node startup and for maintaining synchronization of the estimated time over the lifespan of the node. The clock synchronization is maintained by comparing the expected arrival time of the block, which is calculated based on the block generation interval, with the local clock when the block is received, with a slight correction in the local clock in the direction of the discrepancy. This protocol also suggests that the timing of block broadcasts be adjusted to ensure that the estimated time does not deviate too far from those of real-world clocks. However, because this adjustment is based on only one piece of time information per block, the number of samples is considered to be small. In addition, it is unclear to what extent the local clock is modified during the adjustment; therefore, this topic is not considered in the comparison in this paper.

III. Proposed Method

In this section, we propose a method that considers the effects of the block propagation time, clock drift, and other factors that have not been fully considered in existing methods, as described in Section II. Similar to the existing methods [5], [8], the proposed method estimates the offset between the clock of the block producer and its own local clock based on the received time of the blocks and the information in the block for blocks received during a certain period. In the proposed method, instead of using the median of the estimated offsets as the modification value, the block propagation time is estimated according to the median of the estimated offsets up to the current point; thus, the local clock does not deviate from the real-world clock during each adjustment.

The synchronization accuracy is also improved by taking the drift and frequency of the clock adjustment into account.

A. Calculation of the Estimated Offset

To estimate the offset between the block producer and the local clock, we use the slot number i stored in the block and the value of the local clock τ_i recorded when the corresponding block is received. Note that the block propagation time and clock drift are not considered here, as we are describing the offset estimation method in existing methods. Let δ_i be the estimated offset and L_s be the length of the slot. δ_i can be calculated as follows:

$$\delta_i = \tau_i - L_s * i$$

The proposed method calculates the estimated offset δ_i for all blocks received during a certain period of time. The median of the estimated offsets is recorded, and the block propagation time is estimated according to the medians, which are updated after each adjustment.

When the offsets are estimated with this method, we assume that the block is generated and broadcast at the beginning of the slot; thus, we do not consider when or how long it takes to generate the block.

B. Consideration of the Block Propagation Time

The estimated offset can be expressed as follows, where the block propagation time due to the block producer is Δ and the offset between the block producer and the local clock is o.

$$\delta = \Delta + o$$

The offset o can be expressed as follows, where ϵ is the adjustment error of the previous adjustment, c_s is the number of slots that have elapsed since the last adjustment, and d is the clock drift per second of the local clock.

$$o = \epsilon + L_s * c_s * d$$

Thus, the estimated offset δ can be expressed as follows.

$$\delta = \Delta + \epsilon + L_s * c_s * d \tag{1}$$

To estimate the block propagation time, we assume two conditions:

- The clock drift is negligibly small relative to the propagation time.
- If the network is sufficiently synchronized with respect to the average time, the sum of the adjustment errors converges to the initial gap with respect to the average time.

Here, the average time is the average value of the local clocks of all nodes participating in the protocol, and the initial gap is the difference between the average time and the local clock at the start of the protocol. If the initial gap is small, based on Equation (1) and the above assumptions, the average of the median of the estimated offsets described in Section III-A can be approximated as the average block propagation time. We call this the estimated block propagation time. The estimated block propagation time is reflected in the adjustment to suppress deviations between the clock and the real-world clock.

In this case, the modification value $adjust_\tau$ can be expressed as follows, where median(δ) is the median estimated offset recorded during the relevant period and Δ_{est} is the estimated block propagation time.

$$adjust_\tau = \text{median}(\delta) - \Delta_{est}$$

The modification values are recorded for each adjustment, and these values are used to estimate the drift.

For the second assumption, one method for determining if the local clock is sufficiently synchronized with the average time of the network is to verify that the medians of the most recent estimated offsets converge to a certain value. However, since the block propagation time cannot be estimated until the clock synchronization has been confirmed, the average time deviates from that of the real-world clock due to the clock adjustment that occurs during this time, as is the case with the existing methods.

TABLE I: Parameter settings.

# of nodes	100
# of peers	5
Initial clock gap	Variable
Delay	$\simeq$ 2.0[s] per hop
Slot size	12 [s]
# of slots per epoch	32
Tick rate	0.1 [s] per tick
Drift	$\leq$ 8.6 [s] per day
# of ticks	240000

C. Consideration of Drift

Even if the local clock is successfully adjusted, the clock drifts during the adjustment interval unless the clock is corrected in advance. In the proposed method, the previous modification values are recorded, as well as the number of slots between the time of the previous adjustment and the time when the corresponding block used to calculate the modification value was generated, and both values are used to estimate the drift.

Based on the previous assumptions, the sum of the adjusted values can be approximated as the sum of the deviations caused by the drift. Let k be the total number of slots recorded and sum($adjust_\tau$) be the sum of the modification values, which can be expressed as follows:

$$\text{sum}(adjust_\tau) = L_s * k * d$$

Therefore, the estimated drift can be calculated based on the recorded values, and the calculated estimated drift can be used to determine the amount of drift per tick.

D. Frequency of Clock Adjustment

In the existing methods, the timing of the adjustment varied between each epoch, block, and tick. If the time information used for the adjustment is reset during each adjustment, the frequency of adjustment should also be considered, since the amount of time information used for adjustment changes depending on the timing of the adjustment. In the proposed method, the timing of the adjustment is determined at the end of each epoch, as in [8], and the frequency of the clock adjustment is modified by defining a threshold based on the amount of time information required for the adjustment. When the amount of time information exceeds this threshold value, the local clock is corrected at the end of the epoch, and the recorded time information is deleted; otherwise, no action is taken. The number of epochs is not changed to modify the frequency of clock adjustment because it cannot be guaranteed that a certain amount of time information is collected during an epoch if the local clock values are widely distributed.

IV. Experimental Settings

This section describes the settings for the experiments and simulations.

A. Simulator

We implemented the simulator for the experiments based on the simulator used in [5]. This simulator was designed for Ethereum 2.0, and for the experiments in this paper, Ethereum 2.0 was assumed to be a example of the PoS blockchain. In the simulator, each node was assigned an initial value for its local clock based on the values in Table I and advanced its local clock based on the tick rates in Table I. The initial value of the local clock of an ideal node was assumed to be 0. Each node recognized a round based on the value of its local clock and performed its assigned role during that round.

B. Network Model

The network model used in the experiments was based on the underlying simulator configuration. At the start of the protocol, the number of neighbors listed in Table I were randomly selected, and the node communicated with these neighbors. The message propagation time was based on the values in Table I, and the received messages were broadcast to all neighboring nodes.

C. Consensus Model

As described in Section II-A, Ethereum 2.0 uses rounds known as epochs and slots. Each epoch and slot uses the values adopted in Ethereum 2.0, which are shown in Table I. Each node is assigned a block generation role for each slot, and the node generates a block at the start of the slot and broadcasts that block to the network. For block generation, the parent block is determined by a fork-choice rule known as LMD GHOST, and the node proposes its own block as a child block of the parent block. The voting message used to calculate the score in LMD GHOST is generated once per epoch by all nodes.

D. Drift

Since the underlying simulator does not have a mechanism for simulating the drift of the local clock of each node, we determined the drift with a simple method. According to [10], an RTC with a typical crystal oscillator drifts by up to 8.6 seconds per day. The simulator replicates this behavior by assigning each node a uniform daily drift value between $[0.0, 8.6]$ at the start of the execution. The assigned values are converted to per tick values and added to the tick rate values at each tick, as shown in Table I. In this paper, the drift values are assigned uniformly; however, an exact distribution will be considered in future work.

E. Comparison Methods

In this section, we describe the methods that were compared with the proposed method. The notations are summarized in Table 2. The comparison methods were selected based on the condition that no additional communication is required to accomplish clock synchronization.

NAT is used as implemented in the underlying simulator and adjusted using the estimated offsets of all neighboring nodes recorded at each tick. For RTP, since the valid, finalized blocks cannot be obtained during the relevant epoch under this experimental setting, they are replaced by all blocks received during the relevant epoch for comparison. The offsets of these methods are calculated using the method in Section III-A, and the median of the offsets is used as the modification value. proposed w/ prop time includes the estimated block propagation time of our proposed method for RTP. proposed w/ drift adds the estimated drift to proposed w/ prop time. proposed w/ drift ($t = 100$) modifies the clock adjustment frequency based on the amount t of time information required for adjustment in proposed w/ drift. In this paper, the experiments were conducted with $t = 100$.

TABLE II: Notation of the clock synchronization methods in this paper.

noadjust	No adjustment
NAT	[5]
RTP	[8]
proposed w/ prop time	Proposed method: only the block and propagation time are considered
proposed w/ drift	Proposed: drift is also considered
proposed w/ drift ($t = 100$)	Proposed: frequency of clock adjustment based on threshold t is also considered

Since there is a risk of introducing significant erroneous corrections, as described in Section III-C, the drift correction is performed by dividing the estimated drift value by 10^4. The frequency of drift correction is determined after the local clock has been adjusted five times.

F. Evaluation Metrics

The gap between the maximum and minimum local clocks of all nodes in the network at the time of recording is used to assess the synchronization accuracy. A recording is taken when all nodes in the network have completed their nth adjustment, where $n \in \{0, 1, ...\}$. However, the recording is taken at the end of each epoch for NAT since this method performs an adjustment for each tick.

As another evaluation metric, to evaluate the difference with the real-world clock, we record the difference between the average value of the local clocks of all nodes and the execution time recorded at the same timing as the abovementioned gap.

V. Experimental Results

In this section, we evaluate the proposed method by comparing it with existing methods based on the settings described in Section IV.

A. Without an Initial Gap

First, we discuss the results when the initial gaps of all local clocks are zero. The experiments with initial gaps of zero are intended to confirm the effect of the block propagation time on each method. Furthermore, we confirm the effectiveness of the proposed method, which is based on an assumption of sufficient synchronization.

Figures 2 and 3 show the results of the gap between the maximum and minimum local clocks and the gap between the average of the local clocks and the real-world clock,

respectively. Figure 2 shows that the proposed method synchronizes with higher accuracy than the existing methods. The gap between the maximum and minimum local clocks is smaller when using proposed w/ drift than when using proposed w/ prop time, which confirms the expected effect of the drift estimation. However, the results are not considerably pronounced because the correction value of each tick is small, as described in Section III-C. For proposed w/ drift ($t = 100$), synchronization is achieved with the highest accuracy and the fewest adjustments during execution, confirming that reducing the adjustment frequency to ensure a certain number of time information.

Figure 3 shows that the average value of the local clocks of all nodes in the network deviates more from the value of the real-world clock with increasing time when using the existing methods. In contrast, with the proposed method, the transition is closer to the real-world clock, which confirms the effectiveness of the block propagation time estimation. Table III shows the average value of the local clocks of all nodes at the end of the execution. In the table, "ideal" indicates the real-world clock, i.e., the execution time in the simulation. The results show that the proposed method completes the execution in a time that is sufficiently close to that of the real-world clock. The results of proposed w/ drift and proposed w/ drift ($t = 100$) deviate more from the real-world clock than the results of proposed w/ prop time because the proposed method corrects the drift estimation according to the average tick rate in the network.

B. With an Initial Gap

Next, we describe the results when an initial gap between $[-2.0, 2.0]$ is uniformly assigned to the local clocks of all nodes at the start of execution. The proposed method in this experiment adjusts the local clocks in the same manner as RTP until a constant synchronization is confirmed to satisfy the second assumption in Section III-B, i.e., that the local clock is synchronized to the network. After synchronization, the block propagation time and drift are estimated for each method. The synchronization can be confirmed with the method described in Section III-B; however, for simplicity, we assume that the five synchronization adjustments are performed using the same adjustment method as for RTP. To improve the accuracy of the block propagation time estimate, the block propagation time is not estimated until the 10th adjustment, when synchronization can be assumed, and these modification values are recorded and used for the subsequent block propagation time estimation.

Figures 4 and 5 show the results of the gap between the maximum and minimum local clocks and the gap between the average of the local clocks and the real-world clock, respectively. Table IV shows the average value of the local clocks of all nodes at the end of execution. Figure 4 shows that the gaps between the maximum and minimum local clocks using proposed w/prop time and proposed w/drift are equivalent to the gaps obtained using the existing methods. As with the results in Section V-A, the effect of the drift estimation on the proposed method is not clear with this result;

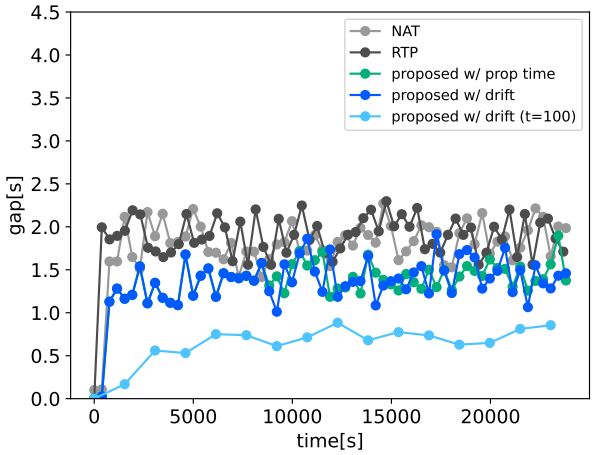

Fig. 2: Gap between the maximum and minimum local clocks (without an initial gap).

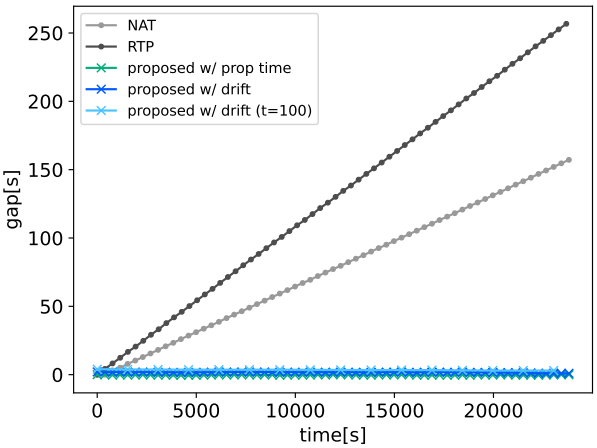

Fig. 3: Gap between the average of the local clocks and the real-world clock (without an initial gap).

TABLE III: Average local clock value at the end of execution (without an initial gap).

ideal	24000.00
noadjust	24001.13
NAT	23841.57
RTP	23743.20
proposed w/ prop time	24000.07
proposed w/ drift	24000.94
proposed w/ drift ($t = 100$)	24001.02

thus, additional experiments and more detailed analyses are needed. It can be confirmed that proposed w/ drift ($t = 100$) achieves the most accurate synchronization in many cases, similar to the no initial gap scenario. Figure 5 and Table IV show that the proposed method transitions closer to the real-world clock than the existing methods, confirming the effect of the block propagation time estimation. The gap with respect

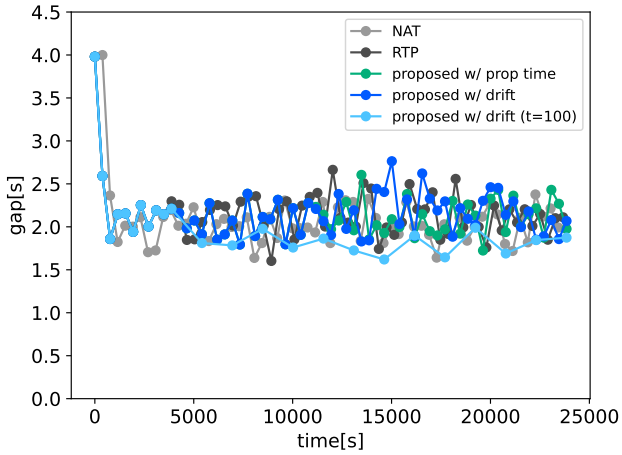

Fig. 4: Gap between the maximum and minimum local clocks (with an initial gap).

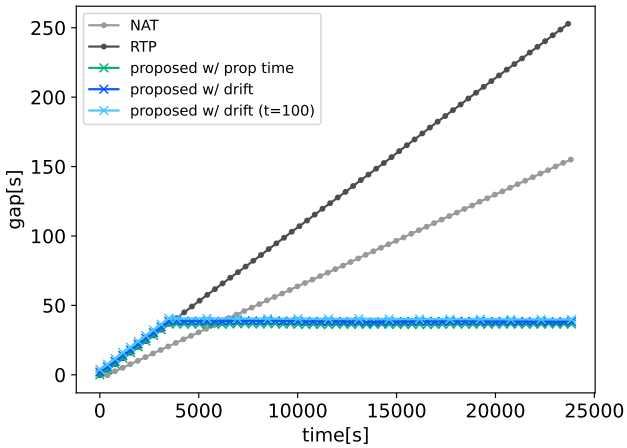

Fig. 5: Gap between the average of the local clocks and the real-world clock (with an initial gap).

TABLE IV: Average local clock value at the end of execution (with an initial gap).

ideal	24000.00
noadjust	24001.45
NAT	23843.58
RTP	23747.16
proposed w/ prop time	23963.45
proposed w/ drift	23963.72
proposed w/ drift ($t = 100$)	23964.07

to the real-world clock is not zero in the proposed method because the local clock is adjusted as in RTP to synchronize the local clock with the network. However, the proposed method can suppress the effect of the block propagation time on the gap with the real-world clock.

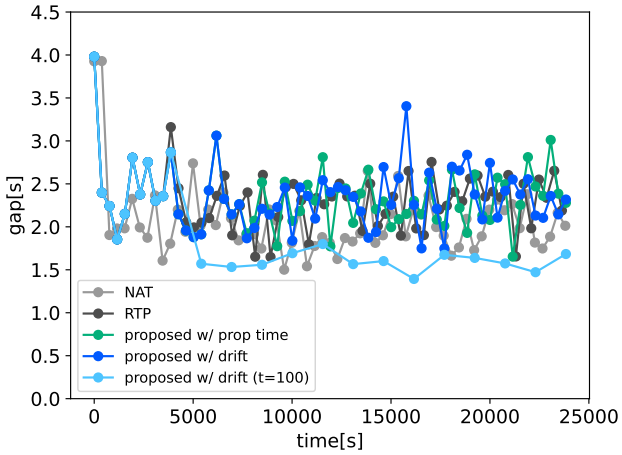

Fig. 6: Gap between the maximum and minimum local clocks (for nodes that do not follow the protocol).

TABLE V: Average local clock value at the end of execution (for nodes that do not follow the protocol).

ideal	24000.00
noadjust	24001.45
NAT	23916.46
RTP	23847.05
proposed w/ prop time	23978.85
proposed w/ drift	23978.78
proposed w/ drift ($t = 100$)	23978.48

C. Nodes that Do Not Follow the Protocol

Finally, we conduct an experiment in which there are nodes that do not follow the clock synchronization protocol. As an example of a node that does not follow the clock synchronization protocol, we assume the existence of a node that does not adjust its local clock. In this paper, we present results for a case in which 30% of the nodes in the network do not adjust their clocks during the execution. The evaluation metric for this result considers only the local clocks of the honest nodes that adjust their clocks rather than the local clocks of all nodes.

Figure 6 and Table V show the results of the gap between the maximum and minimum local clocks and the average value of the local clocks of all nodes at the end of execution, respectively. Figure 6 shows that all methods except proposed w/ drift ($t = 100$) have comparable accuracies, similar to the results in Section V-B. Furthermore, compared to the results of Section V-B, nodes that do not adjust their clocks reduce the clock synchronization accuracy in the network of honest nodes. proposed w/ drift ($t = 100$) achieves the highest synchronization accuracy in many cases, similar to the results in Section V-B, since the lower bound of the time information for the adjustment is guaranteed. Table V confirms that the proposed method can synchronize with a time that is closer to that of the real-world clock than the existing methods, similar

to the results in Sections V-A and V-B.

VI. Conclusion

In this paper, we propose a new clock synchronization protocol for slot-based PoS blockchains that is independent of external protocols. The proposed method effectively uses the received block time and the information within the block to suppress the effects of the block propagation time and drift, which have not been fully considered in existing methods. In addition, the frequency of clock adjustment can be changed to improve the synchronization accuracy. Simulation experiments show that the proposed method can synchronize with a time that is closer to that of the real-world clock and with higher accuracy than existing methods. We also confirmed that this effect can be observed even when there are nodes that do not follow the protocol.

Future work will include experiments that assume the presence of malicious nodes.

References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Decentralized Business Review*, p. 21260, 2008.

[2] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, no. 2014, pp. 1–32, 2014.

[3] D. Mills, J. Martin, J. Burbank, and W. Kasch, "Network time protocol version 4: Protocol and algorithms specification," 2010.

[4] E. Foundation, "Go Ethereum," https://geth.ethereum.org/.

[5] V. Buterin, "Network-adjusted timestamps," 2018, https://ethresear.ch/t/network-adjusted-timestamps/4187.

[6] A. Vlasov, "Lightweight Clock Sync Protocol for beacon chain," 2020, https://ethresear.ch/t/lightweight-clock-sync-protocol-for-beacon-chain/7307.

[7] BloodyRookie, gimre, and Jaguar0625, "Symbol from NEM - Technical Reference," Web document, Jan. 2020. [Online]. Available: https://symbol.github.io/symbol-technicalref/main.pdf

[8] H. K. Alper, "Network time with a consensus on clock," *Cryptology ePrint Archive*, 2019.

[9] A. Hartl, T. Zseby, and J. Fabini, "BeaconBlocks: Augmenting Proof-of-Stake with On-chain Time Synchronization," in *2019 IEEE International Conference on Blockchain (Blockchain)*. IEEE, 2019, pp. 353–360.

[10] Wikipedia, "Real-time-clock," https://en.wikipedia.org/wiki/Real-time-clock.