Blockchain, what time is it? Trustless Datetime Synchronization for IoT

Emanuel Regnath
emanuel.regnath@tum.de
Technical University of Munich, Germany

Nitin Shivaraman
nitin.shivaraman@tum-create.edu.sg
TUMCREATE, Singapore

Shanker Shreejith
shankers@tcd.ie
Trinity College Dublin, Ireland

Arvind Easwaran
arvinde@ntu.edu.sg
Nanyang Technological University, Singapore

Sebastian Steinhorst
sebastian.steinhorst@tum.de
Technical University of Munich, Germany

***Abstract*—Time synchronization among IoT devices is a fundamental requirement for efficient and reliable communication on a global scale. Common synchronization schemes such as NTP operate on a trust-based client-server model, which does not scale well in a decentralized network because single server failures can lead to a severe downtime before re-establishing synchronization. Public blockchains such as Ethereum provide a trustless network and tamper-proof time-stamped data that is freely available.**

In this paper, we leverage the availability of time information in the block headers, which are very small (several hundreds of bytes) compared to the full blocks and can be validated without participation in the mining process. Our approach uses two estimators that are fed with the timestamps from block headers as well as the elapsed time between consecutive block receptions to estimate the true time to an accuracy of one second.

We evaluate our approach by extensive validation on blockchain data from different geographical locations across the globe and show that global synchronization can be established despite the non-deterministic behavior of blockchains such as mining difficulty, network latencies and forks.

***Index Terms*—Time Synchronization, NTP, Hash, Blockchain**

I. Introduction

Distributed IoT devices use time synchronization to a global reference time, such as UTC, to agree on communication periods, schedule actions and establish an order of registered events. Many applications such as traffic signaling systems [1] or logging events from sensor measurements, require accuracy from a few hundred milliseconds to a second.

Existing time synchronization methods, such as the Network Time Protocol (NTP) [2], rely on the questionable assumption that certain servers can be trusted. In order to authenticate these servers, digital certificates need to be verified, which requires more computational power than many embedded devices can afford. Furthermore, in case of malicious NTP servers, the time to synchronize is severely affected [3]. This is further aggravated if the gateway is malicious since fallacious time is received independent of the chosen server. Since communication over the Internet involves malicious and faulty nodes, the NTP synchronization can not provide robust and secure timing information for embedded devices.

Consequently, a mechanism is needed that 1. does not rely on individual servers, 2. provides verifiable timing data, and 3. uses only light-weight cryptography that can be computed by resource-constrained devices. We thus aim to achieve secure datetime synchronization for a decentralized network in the presence of malicious nodes.

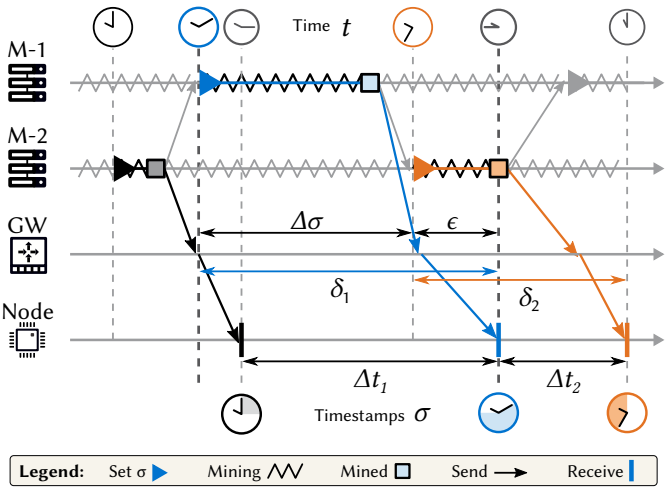

Figure 1: Synchronization scenario. A node receives blocks with timestamps σ_i from two miners M-1 and M-2 via a gateway GW. Upon reception, $\sigma_i = t_i - \delta_i$ because each miner sets the timestamp when it receives the *previous* block. The goal is to estimate t based on σ_i and the observable Δt between receptions.

Blockchain provides a *trustless* system for agreement on a global scale. Each accepted block in the chain is immutable and reinforced by every new block appended to the chain. In case of Proof-of-Work (PoW) chains, even the content can be verified for integrity due to inherent properties of its hash value. This allows secure verification of any block with a minimum amount of hash operations, which is feasible on most embedded devices [4].

In this paper, we extract the timestamps (datetime information) from block headers of the Ethereum blockchain to estimate the current time. Note that we only need to read the headers of *already mined blocks* and are not required to participate in the mining process in any sense. This passive listening approach enables any device with Internet connection to access and verify the information from any public PoW blockchain. As shown in Figure 1, upon reception of a valid block in a typical blockchain system, each miner will create a new unconfirmed block and starts mining. If the mining is successful, the miner will distribute its new block, which contains a timestamp corresponding to the creation time. The objective is to enable each device to synchronize to UTC clock independently using the timestamp information in the block headers.

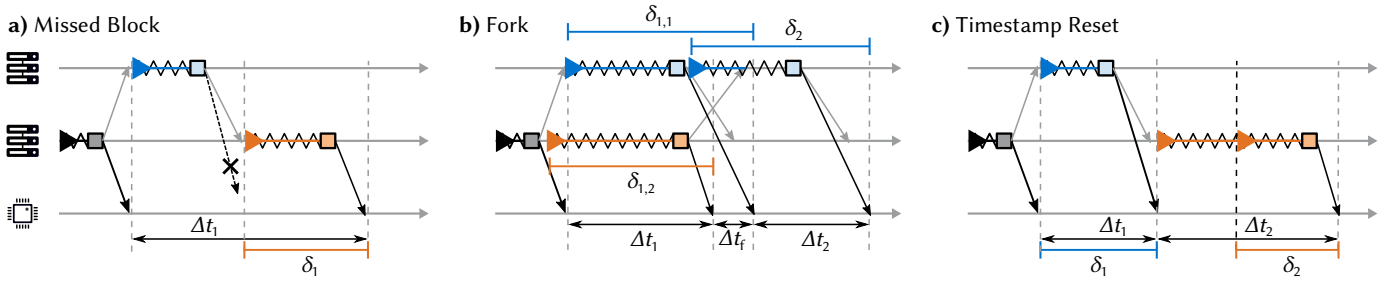

Figure 2: Special cases when syncing via Blockchain: a) A missed block occurs when a received block is further in the future than the next block. b) A fork occurs when two valid blocks with the same index i are received. c) A timestamp reset occurs when miner M-2 updates the block timestamp during mining which leads to the condition $\delta_i < \Delta t_i$.

Our evaluation in Section III shows that the timestamps σ and the elapsed time between block receptions Δt are sufficient to estimate the current UTC to an accuracy of one second. However, this is non-trivial due to issues such as timestamp resets, missing blocks, and forks in the blockchain that will be discussed in this paper.

A. Contributions

We propose a secure, decentralized, and trustless synchronization scheme for embedded IoT networks. Our novel method involves an IoT node that passively listens to the latest blocks of a public blockchain that uses Proof-of-Work (PoW). In particular, we

- investigate the delay behavior of blockchain headers from which we derive a formal model (Section III),
- propose a novel synchronization scheme using only blockchain headers (Section IV),
- analyze convergence, accuracy, drift, and security (Section VI) of our scheme.

By using only block headers, we drastically reduce the communication overhead while enabling even highly resource-constrained devices to receive, process, and securely validate the timing information. Furthermore, we utilize the already spent energy by the miners on the block mining instead of creating new cryptographic proofs.

II. Background

A. Blockchain

The blockchain is a distributed data structure that stores a system state over time and is shared and replicated by all nodes [5]. Formally, a blockchain is an ordered chain

$$\mathcal{C} = \{B_i \| i \in 1, \ldots, n\}, \qquad B_i \prec B_{i+1} \tag{1}$$

of n blocks B_i where n is the height of $\mathcal{C}$ and i the height or index of B_i. Each block confirms and reinforces the data of its preceding block. By including the previous block hash h_p in each block, the integrity and order of the blocks is secured because any change to the data of an existing block would result in changing hashes of all blocks. A block header with a timestamp σ_i is represented by the tuple $B_i = \langle h_\mathrm{p}, \sigma_i, h_\mathrm{d} \rangle$. The actual data D_i of a block can be of arbitrary structure and is bound to a block only by its hash digest $h_\mathrm{d} = H(D_i)$, where $H(\cdot)$ is the hash function.

B. Proof-of-Work (PoW)

Most blockchain implementations use Proof-of-Work (PoW) for block creation and our approach requires PoW for block verification. Proof-of-Work involves *mining* a special hash value for new blocks with the data and the hash of the previous block. The time required for mining is called *Blocktime* and could vary significantly as it depends on the difficulty of the hash-value limit set by the blockchain [6]. Propagation delay is the time for transmitting the block to the rest of the network, which varies in different parts of the network with some nodes receiving the block faster than others [7]. Table I gives an overview of three prominent blockchains in use with information on their blocktime, height, hash rates and difficulty. Difficulty represents the minimum value of the Proof-of-Work a blockchain must have, and is represented as a bit mask to ensure the Proof-of-Work solution satisfies this condition. Difficulty varies with blockchain and higher difficulty translates to higher energy expenditure and higher security against tampering attacks.

Bitcoin is the first and oldest implementation of blockchain. It has the highest hash rate of 90 Exa hashes (SHA-256) per second and the longest average blocktime of 10 minutes.

Ethereum has a hash rate of 180 Tera hashes (SHA-256) per second and the fastest average blocktime of 15 seconds among the three.

Litecoin is the newest blockchain of the group, has 2 minutes blocktime, and uses a hash function called *S-crypt*.

For our approach, we use the Ethereum blockchain to estimate datetime for a network of IoT nodes as it has the lowest average blocktime.

III. Blockchain Timing Model

In this section, we will measure and evaluate the timestamp information in the Ethereum blockchain. We derive assumptions from our observations, which has been used to create our model of timestamp distribution.

A. Time Notation

It is important to understand that we distinguish several types of time variables of a block B_i sent by any miner M to our IoT node:

1) The true, absolute and unknown time $t \in \mathbb{R}$.
2) The elapsed time $\Delta t = t_i - t_{i-1}$ in seconds between the reception of blocks B_{i-1} and B_i that is, in general, observable by the internal clock of any node.

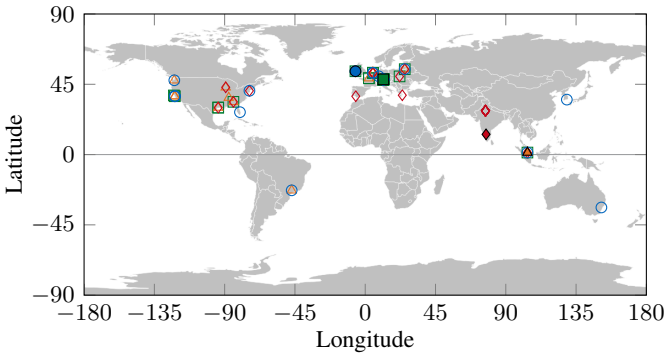

Figure 3: True locations (filled) of our geth clients and estimated locations of connected peers based on their IP address (outline).

3) The estimated time $\hat{t}_i$ of the true time t_i in epoch seconds.
4) The observed timestamp σ_i in epoch seconds, which is stored in block B_i.
5) The timestamp delay $\delta_i = \sigma_i - t_i$ consisting of the mining time of miner M and the transmission delay to our node.

B. Observation of Block Timestamps

First, we recorded a total of 5100 blocks from Ethereum and analyzed the statistical distribution. We used the `geth` client version 1.8.23 and executed the following capture script.

```
eth.filter('latest', function(error, hash){
  t_localtime = Date.now() / 1000;
  block = eth.getBlock(eth.blockNumber)
  if(block.hash == hash) {
    console.log([eth.blockNumber,
      t_localtime, block.timestamp].join(",␣"));
  }});
```

We have performed 8 capture rounds of blockchain data from the Ethereum MainNet at four geographical locations: Munich-Germany, Singapore-Singapore, Bangalore-India and Dublin-Ireland. As the map in Figure 3 shows, the connected peers are not locally clustered but distributed among the large cities around the globe.

C. Challenges when reading Blockchain Timestamps

While the block headers allow global time to be extracted and used, there are numerous challenges associated with the timestamp delays that need to be addressed and resolved.

a) Gaps A gap occurs when a client misses one or several blocks due to the timeout or disconnection of a peer. As shown in Figure 2a, the IoT device does not receive the block from Miner 1 but only the succeeding block from Miner 2. This results in a time difference between two consecutive blocks (Δt_1) greater than the block time (δ_1) of the recent block leading to a positive offset. Since Δt_1 does not reflect the true delay, it should not be used for estimation. Instead, the IoT device needs to extrapolate the current time estimate from its previous estimation.

b) Forks A fork happens when two *different* blocks are mined on top of the *same* previous block. In this case, there exist two different versions of the blockchain. The network is initially unsure of which block will be included in the final blockchain and will, therefore, buffer a number of blocks to ascertain which chain will be accepted and continued. An example of a Fork is shown in Figure 2b, where both miners 1 and 2 create a valid block. However, the peers accept only one of the blocks and include it in the blockchain. The other block is discarded. Depending on which block is accepted, the elapsed time between the forked blocks Δt_f needs to be added either to Δt_1 or Δt_2.

c) Timestamp Reset The assumption that the timestamp σ_i of a block $\mathcal{B}_i$ corresponds to the time when the miner received the previous block $\mathcal{B}_{i-1}$ might not hold. Each miner may, during mining, reset the timestamp to the current time, which is illustrated in Figure 2c. Miner 2 starts mining the block after receiving the block from miner 1. However, the timestamp is reset during the mining process by miner 2 and this is received by the IoT device when the mined block is propagated. This phenomenon results in timestamps being faster than the block reception time similar to gaps, i.e., $\delta_2 < \Delta t_2$, yielding a positive offset.

Name	Blocktime	Height	Hashrate (hash/s)	Difficulty
Bitcoin	10 min	5.96 k	90 E	12.75 T
Ethereum	15 sec	8.63 M	180 T	2.75 P
Litecoin	2 min	1.71 M	324 T	11 M

Table I: Common blockchains and their parameters (End 2019)

D. Observation Results

Our results are given in Figure 4. We found that on average $\Delta t \approx 14.757$ s and $\Delta\sigma \approx 14.743$ s. These values are close to the theoretical blocktime of 15 s. The small difference confirms the fact that miners set the timestamp of a new unmined block immediately after they receive the previous mined block. The true block delay δ, which we measured on reception against the NTP synced system time, is on average larger than $\Delta\sigma$. This indicates that miners (who set timestamps on reception) receive the blocks faster than our geth client. This is reasonable given the high-speed networks, as miners choose reliable and low-latency peers in large mining pools to gain an advantage over other miners in the mining race.

As expected, we found the timestamps of the blocks to be monotonously increasing except when forks occur. A key observation from the timestamp delays (δ_i) shown in the Figure 4) is that they follow an exponential distribution, aligning with prior observations in literature [7]. Among all 5100 measured blocks, we observed 93 forks, 327 gaps and an estimated number of 225 timestamp resets. While forks and gaps are directly measurable from the block index, the timestamp resets are more difficult to identify because they could occur any time and a positive offset could also be caused by receiving the previous block before the miner. Since this is unlikely for large offsets, we found 0.5 s to be a reasonable decision threshold before we consider a positive offset to indicate a reset.

E. Modelling Timestamp Distribution

Based on our observations, we assume that

- Miners will normally set the timestamp of a new block to be mined at the moment they receive a valid block.
- Miners will reset the timestamp during mining in roughly 4 % of blocks.

	mean [s]	stdev
Δt	14.757	16.82
$\Delta\sigma$	14.743	16.82
δ	15.028	12.88

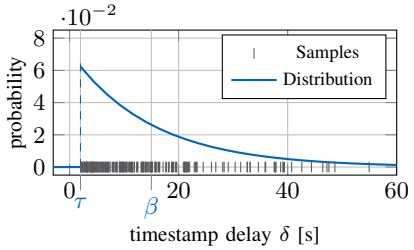

Figure 4: Left: Results of the time stamp analysis given in seconds. Right: Fitted probability density for the timestamp delay of the Ethereum Blockchain with $\beta = 15\,\mathrm{s}$.

- Miners are better inter-connected and will receive most new blocks earlier than light-clients. Thus $\Delta t < \delta$ for most blocks.

As observed from our initial experiments, the probability density function (pdf) of the observed timestamp delays of PoW blockchain follows an exponential distribution that is expressed as

$$p(\delta, \beta, \tau) = \begin{cases} 0 & \delta < \tau, \\ \frac{1}{\beta} \exp\left(-\frac{\delta - \tau}{\beta}\right) & \delta \geq \tau \end{cases} \tag{2}$$

with the scale parameter β as the average block time in seconds and the shift parameter τ as the minimum timestamp delay. For example, for the Ethereum Blockchain we found $\beta \approx 15\,\mathrm{s}$ and $\tau \approx 1\,\mathrm{s}$ which is plotted in Figure 4.

The distribution of the delays is exponential because of the PoW to create a block is a repeated Bernoulli trial. Each attempt to find a nonce that results in a hash value with enough leading zero bits is one Bernoulli trial.

The exponential distribution describes the probability distribution of the time between events in a Poisson point process. The block creation events follow a Poisson distribution because they are of sum of n independent Bernoulli distributed variables, where n is the number of participants in the network.

For the remainder of this paper, we assume a shifted exponential distribution for the overall timestamp delay δ.

IV. Our Approach

In this section, we introduce two different estimation methods that offer minimal drift in observation even in the presence of block uncertainties. Each method has a naive approach, which we will use to illustrate the idea and an improved variant that can be used for the actual synchronization resulting in a total of four estimators.

A. Maximum Likelihood Estimator (MLE)

The MLE is a general approach to estimate the timestamp delay only based on timestamp observations. The likelihood function of the timestamp delay distribution is

$$L(\tau, \beta) = \frac{1}{\beta^n} \cdot \exp\left(-\frac{n\left(\overline{\delta} - \tau\right)}{\beta}\right) \quad \text{with } \overline{\delta} = \frac{1}{n}\sum_{i=1}^{n} \delta_i$$

where $\overline{\delta}$ is the mean of the timestamp observations δ_i and and n the total number of the observations. With respect to τ, the function increases until the minimum observed offset $\delta_{\min}$ and thus, we obtain $\hat{\tau}_{\mathrm{MLE}} = \delta_{\min}$. We can use this result to formulate a simple decision rule for estimating the current time based on the received timestamps. Whenever we receive a timestamp σ_i that is further in the future than our prediction $\hat{t}_i = \hat{t}_{i-1} + \Delta t$, based on previous timestamps, we will switch to this new time $\hat{t}_i = \sigma_i$. This estimation works because the timestamps are always delayed but from time to time we will receive a timestamp with a lower delay than all previous timestamps.

This estimator will not overshoot the true time t and therefore serves as a lower bound for our estimation $\hat{t}$. However, the MLE has two drawbacks. First, it converges slowly towards the true time and it will not get closer than τ.

Second, since we accept the timestamp from the latest block, an attacker *could* mine one fake block with a timestamp in the future, which would result in a permanently wrong estimation.

B. Secure Lower Bound Estimator – LowerSec

We now introduce our first proper estimator *LowerSec*, which is the improved variant of the MLE. To overcome the second limitation of the MLE, we accept a timestamp from a block B_i only if m additional blocks have been mined on top, which reinforces B_i. This is the same technique used to consider a transaction settled and provides increasing security with the number of additional blocks m. For our secure lower bound estimator *LowerSec*, we have chosen $m = 10$.

C. Time Difference Estimator – TimeDiff

While MLE-based approaches only provide for a lower bound, we introduce *TimeDiff* estimator to improve upon the accuracy of time estimation. Remember that the correct time t could be obtained as $t_i = \sigma_i + \delta_i$.

However, we cannot observe δ_i directly and need to estimate it based on the available information. Basically, $\delta_i = \delta_{m,i} + \tau_i + \epsilon_i$ where $\delta_{m,i}$ is the mining time, τ_i the propagation delay to the next miner, and ϵ_i the additional propagation delay to our node. Since mining is a race between miners, we assume that miners use a fast hardware and low latency connection, such that the time between receiving a block and setting the timestamp of the next pending block is negligible. This means that $\Delta\sigma_i = \sigma_i - \sigma_{i-1} = \delta_{m,i} + \tau_i$, which allows us to observe the timestamp delay between miners.

Furthermore, we assume that miners receive blocks earlier than our node, which means $\epsilon_i > 0$. Again, ϵ_i cannot be observed directly, but since $\Delta t_i = \epsilon_{i-1} + \delta_i$ and from our observations we know that $\sum \Delta t_i \approx \sum \delta_i$, we can conclude that ϵ_i is usually small.

Therefore, we can estimate the true time t with

$$\hat{t}_i = \sigma_i + \Delta t_i + Q_{\mathrm{e}} \tag{3}$$

where $Q_{\mathrm{e}} = 0.5\,\mathrm{s}$ is the quantization error for timestamps. However, *TimeDiff* produces a lot of peaks due to the issues explained in Section III-C and overshoots the true time.

D. Peak Median Kalman Filter – PMK

To overcome the peak problem of TimeDiff, we propose a Kalman filter estimator with the previous highest peaks which runs the median of them. The median of the peaks is fed to the Kalman Filter for the estimation. This ensures that we get a secure and accurate estimation near zero. Using the median

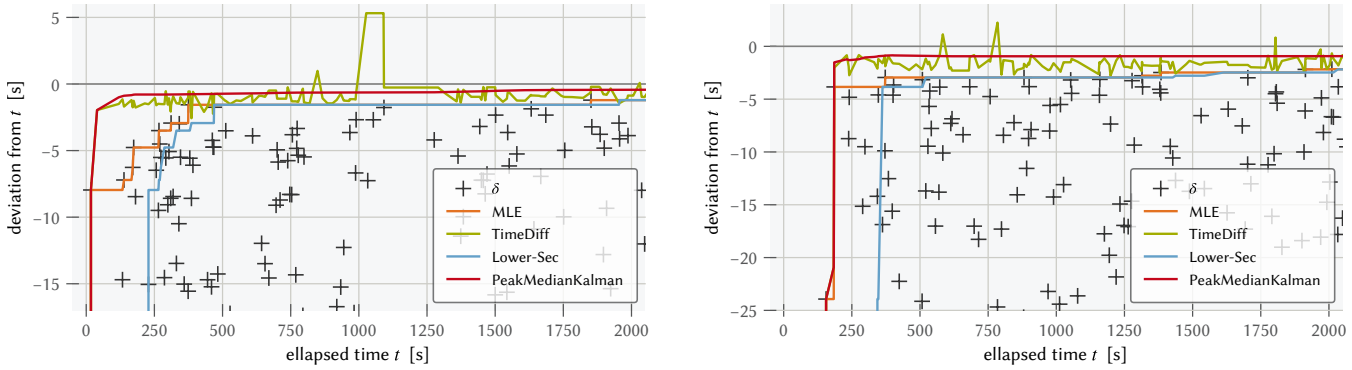

Figure 5: Comparison of all estimators for two capture rounds in Ireland (left) and Singapore (right). For simplicity, we plot the deviation from the true time t such that t corresponds to the x-axis. δ are the true timestamp delays and only shown for reference. The MLE and its variant LowerSec serve as a very secure lower bound. The TimeDiff estimator is closer to t but severely overshoots when a miner resets a timestamp, which can be seen at $t \approx 1000$ s on the left. Due to filtering, the PeakMedianKalman estimator converges fast and without overshooting. As seen on the right plot, there can be an initial delay if the node cannot find enough peers from the beginning.

among the peaks ensures that a few overshooting peaks are filtered out. Even if there was an accepted overshot peak, the estimation shifts back to the correct value when peaks stabilize. The PeakMedianKalman will converge to the mean of the peaks of the TimeDiff estimations. It not only provides a fast convergence similar to TimeDiff, but also ensures a stable and secure drift.

1) Kalman Filter In order to smoothen spikes, we use a Kalman Filter. The general system equations are given as

$$\hat{t}_n = \hat{t}_{n-1} + \Delta t + \mathcal{W}_{n-1} \qquad s_n = \hat{t}_n + \mathcal{V}_n \tag{4}$$

with the time estimate $\hat{t}$, the observed timestamp s, the elapsed time $\Delta t = t_n - t_{n-1}$, the Gaussian process noise $\mathcal{W}_n$, the Gaussian measurement noise $\mathcal{V}_n$ with the following probability distribution:

$$p(w_n) \sim \mathcal{N}(0, Q_n) \qquad p(v_n) \sim \mathcal{N}(0, R_n) \tag{5}$$

We assume $R \approx 0.3 \cdot 10^{-3}$ as the standard deviation of the measurement noise $\mathcal{V}$, since crystal clocks should be accurate to at least $0.3\,\mathrm{ms}$ over the period between two blocks.

V. Related Work

Time synchronization has been well established in the literature with various goals including energy efficiency, accuracy, speed of convergence and reduction in re-synchronizations.

Most of the notable works in the Wireless Sensor Network (WSN) domain [8], [9] used for embedded IoT devices provide accurate and efficient local clock synchronization without any global datetime information. However, these protocols assume trustworthy data from the nodes which may not be necessarily the case. To handle the presence of malicious devices, [10] uses message authentication codes and secret keys between two pairs of nodes for verification. The approach discussed in [11] models the temporal variation of messages from neighbors and classifies any deviation as an attack. While the issue of authentication is addressed, the security definitions and the codes have to be regularly updated.

Blockchain has gained traction as the decentralized scheme to provide a trustless and secure means of communication. Open Timestamps [12] use timestamp data from bitcoin to only timestamp and validate documents to prove the authencity of a document. Relying on its data structure, [13] used their data to transmit and store the clock data into a ledger verified by a consensus node. It necessitates the presence of a computationally-capable consensus node for verification. The authors in [14] extended the architecture to use Proof-of-Stake (PoS) and a custom blockchain whose length can be controlled. However, the consensus mechanism consumes significant time due to computational complexity and the device of the highest stake has to be re-elected in the event of a failure. In contrast, our approach only requires to passively use specific fields of block headers from freely available public blockchain to achieve datetime synchronization on resource-constrained nodes.

NTP [2] is the most prevalent datetime synchronization protocol in use for network synchronization. NTP relies on a a multi-hop client-server synchronization mechanism, with each level called *stratum* synchronizing to the level above it. GPS or atomic clocks generate pulses at stratum 0 to which stratum 1 synchronizes to a few microseconds. As the stratums increase, the synchronization error grows higher with a longer time to achieve synchronization. Despite its popularity, NTP does not provide a way to securely synchronize the clock. While most time servers available for synchronization are at stratum 3, time servers could synchronize devices from even lower stratums. Since response times vary across these servers, the synchronization time could suffer from delays. Additionally, failure of any such server leads to a routing change to a new server, exacerbating the delay. With package/security updates released once a few months, NTP servers are vulnerable to attacks due to obsolete packages. NTP servers operate on a limited bandwidth and resource, requiring them to limit the sync requests to avoid overload issues [15], leading to longer delays and/or failed sync at clients.

Our approach overcomes these drawbacks of security and cryptographic complexity because the blockchain offers peer-to-peer validation and efficient hash verification. As the blockchain network is inherently decentralized, it can also handle a higher number of requests without any overload issues.

VI. Evaluation

We discuss and evaluate our estimators according to convergence time, accuracy, stability, drift, and security. We experimentally tested and analyzed our estimators by feeding them with the σ and Δt of the previously captured data, which we obtained with the geth client.

a) Convergence, Accuracy, and Stability Figure 5 visualizes these metrics for all estimators for two geographical locations. Table II shows the results when running the estimation over all 8 captured data sets at 4 locations with a total of 5100 blocks.

MLE and *LowerSec* are slowly converging to the current time but are very stable since the error can only decrease. The estimation will never overshoot the true time, and over time these estimators can achieve a fair accuracy of a few seconds.

The *TimeDiff* estimator provides a faster convergence but is also unstable because each estimation is only based on the last two blocks. The *PeakMedianKalman (PMK)* achieves the best synchronization with an average error of $-0.36\,\mathrm{s}$ and a standard deviation of only $0.89\,\mathrm{s}$.

b) Drift Since all estimators use Δt to estimate the time between timestamp inputs, they are equally affected by clock drift. Most internal RC clocks are accurate to 3 %, and therefore, the clock drift for measuring Δt between two Ethereum blocks would be around $0.03 \cdot 15\,\mathrm{s} = 0.45\,\mathrm{s}$ which is reasonable for achieving an overall estimation accuracy within one second. In case of crystal oscillators with an accuracy of $\approx 20 \cdot 10^{-6}$, the drift is completely negligible. However, if a blockchain with a high blocktime is used, measuring Δt with RC oscillators will significantly reduce accuracy.

c) Security The security of each estimator depends on the impact of forged blocks on the estimation. To forge even one block, an attacker would require exorbitant computing power and would still have a low success probability.

While MLE accepts a valid block immediately and could be tricked persistently by one forged block, *LowerSec* is highly secure since it uses only blocks that are confirmed by 10 additional blocks on top. Forging 10 consecutive blocks of any of the blockchains listed in Table I faster than the rest of the network is infeasible and therefore it is infeasible for an attacker to convince *LowerSec* of a fake timestamp.

TimeDiff and *PMK* also accept each block immediately and *TimeDiff* could be temporally tricked by a single forged block. Due to the median filtering, PMK cannot be attacked by individual forged blocks at a low rate. In case an attacker could forge blocks at a constant high rate, PMK would slowly converge to the fake timestamps. However, forging at a high rate is infeasible.

d) Summary We evaluated our approach analytically and experimentally to demonstrate the feasibility of using public blockchains for time-synchronization. Overall, PMK provides the best trade-off between accuracy, convergence and security. It could also be combined with the LowerSec for improved security. For all the approaches presented, we observe that there is no computation and power overhead (due to absence of mining) and negligible memory overhead (storage of 10 timestamps is less than one kilobyte).

	med	avg	stdev	max	tt1s	tt2s
MLE	-1.26	-1.72	1.54	-0.3	12830.0	3846.0
LowerSec	-1.29	-136.65	2740.9	-0.3	13050.0	3966.0
TimeDiff	-0.91	-0.93	1.05	5.3	98.0	35.0
PMK	-0.28	-0.36	0.89	0.6	128.0	32.0

Table II: Estimation errors in seconds given as median, average, standard deviation and maximum. The "Time to 1 s" (tt1s) is the average synchronization time until the error is within ± 1 s. For tt2s accordingly.

VII. Conclusion

We have proposed a novel synchronization method that leverages the public datetime information from the timestamps in block headers. These timestamps are validated and confirmed by a decentralized blockchain network which removes issues such as a trusted relationship and single point of failure found in centralized approaches such as NTP. The timestamps of confirmed blocks alone serve as a very secure lower bound for the estimation of the datetime with a usual accuracy of several tens of seconds. Under the realistic assumption that a node receives blocks not later than 1 s after the other miners on average, our advanced estimator, which uses the Kalman-filtered peaks of $\sigma + \Delta t$, can improve the accuracy to about 1 second. Our approach only requires a node to passively listen to the stream of block headers to provide a secure and robust synchronization with fair accuracy, which makes it suitable for embedded IoT devices.

Acknowledgment: This work was financially supported in part by the Singapore National Research Foundation under its Campus for Research Excellence And Technological Enterprise (CREATE) programme. With the support of the Technische Universität München – Institute for Advanced Study, funded by the German Excellence Initiative and the European Union Seventh Framework Programme under grant agreement n° 291763.

References

[1] U.S. Department of Transportation, "Traffic signal timing manual," No. FHWA-HOP-08-024, Jun. 2008.
[2] D. L. Mills, "Internet time synchronization: the network time protocol," *IEEE Trans. on Communications*, vol. 39, no. 10, Oct. 1991.
[3] J. Burbank, D. Mills, and W. Kasch, "Network time protocol version 4: Protocol and algorithms specification," RFC 5905, Jun. 2010.
[4] E. Regnath and S. Steinhorst, "LeapChain: Efficient Blockchain Verification for Embedded IoT," in *Proc. of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov. 2018.
[5] K. Christidis and M. Devetsikiotis, "Blockchains and Smart Contracts for the Internet of Things," *IEEE Access*, vol. 4, pp. 2292–2303, 2016.
[6] Ethereum Foundation, *Ethereum Block timing analysis*, Jul. 2014.
[7] C. Decker and R. Wattenhofer, "Information propagation in the bitcoin network," in *IEEE P2P 2013 Proceedings*, pp. 1–10, Sep. 2013.
[8] J. Elson, L. Girod, and D. Estrin, "Fine-grained network time synchronization using reference broadcasts," *SIGOPS Oper. Syst. Rev.*, 2002.
[9] M. Lévesque and D. Tipper, "A survey of clock synchronization over packet-switched networks," *IEEE Communications Surveys Tutorials*, vol. 18, no. 4, pp. 2926–2947, Jul. 2016.
[10] S. Ganeriwal, S. Čapkun, C.-C. Han, and M. B. Srivastava, "Secure time synchronization service for sensor networks," in *Proc. of the 4th ACM Workshop on Wireless Security*, pp. 97–106, Sep. 2005.
[11] X. Hu, T. Park, and K. G. Shin, "Attack-tolerant time-synchronization in wireless sensor networks," in *IEEE INFOCOM 2008 - The 27th Conference on Computer Communications*, pp. 41–45, Apr. 2008.
[12] OpenTimestamps, *Open Timestamps*, https://opentimestamps.org.
[13] K. Fan, S. Wang, Y. Ren, K. Yang, Z. Yan, H. Li, and Y. Yang, "Blockchain-based secure time protection scheme in iot," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4671–4679, Jun. 2019.
[14] K. Fan, S. Sun, Z. Yan, Q. Pan, H. Li, and Y. Yang, "A blockchain-based clock synchronization Scheme in IoT," *Future Generation Computer Systems*, vol. 101, pp. 524 – 533, Dec. 2019.
[15] NTP Pool News, *Excessive load on NTP servers*, Dec. 2016.