JOURNAL OF INFORMATION SCIENCE AND ENGINEERING XX, XXXX-XXXX (2024)
DOI: XX.XXX.XXXX.XX

PowerTimestamp: Toward Global Event Ordering*

CHIH-WEN HSUEH+ AND YEN-SHUO CHEN
Graduate Institute of Networking and Multimedia
National Taiwan University
Taipei, 10617 Taiwan
E-mail: {cwhsueh, d11944009}@ntu.edu.tw

In distributed systems, event ordering is a critical issue, but there is still a lack of a complete solution. In 1978, Lamport introduced the concept of partial ordering, which made the construction of event orderings possible based on causal relationships between events. However, it is still unable to determine the chronological precedence for concurrent events, and the logical clock with a different concept of time is not widely accepted. In this paper, we propose a mechanism called PowerTimestamp to address this limitation by constructing a comprehensive ordering for all events in an environment where an event is earlier if it is certain time error ahead of NTP timestamp first or with higher priority such as the estimated computing power. To reach transitivity in total ordering, the time error must be well defined and dynamically adjusted according to the arrival time at the destination environment. PowerTimestamp adopts the General Proof-of-Work model, invented to reach consensus on blockchain, ensuring both trustworthiness and fairness in the establishment of global event ordering for any event arriving in time. With PowerTimestamp, events in distributed systems can be ordered by occurrence and synchronization can be deterministic in time with an error of hundreds of microseconds to hundreds of milliseconds.

Keywords: PowerTimestamp, global event ordering, total ordering, partial ordering, General Proof-of-Work, proof-of-work, NTP.

1. INTRODUCTION

Time, as a concept, remains one of the most enigmatic and foundational elements in human understanding. Despite centuries of philosophical debate and scientific inquiry, a universally accepted definition of time still eludes us. Is time a linear sequence of events, a physical dimension, or merely a construct of human perception? These questions highlight the inherent ambiguity of time, which has far-reaching implications, particularly in distributed systems. In such environments, where multiple nodes operate independently and collaboratively, time is necessary for synchronization, including event ordering, action coordinating, and consistency. However, the ambiguous nature of time introduces significant challenges in achieving these goals.

Received March 11, 2014; revised June 20, 2014; accepted September 28, 2014.
+Corresponding author
*Communicated by the editor.

Event ordering is broadly categorized as total ordering and partial ordering, where total ordering guarantees that all events in the system occur in the same order on all nodes, while partial ordering only guarantees that some events are ordered with respect to each other. It is difficult to define time precisely so as to tell exactly what time is or how time is running. Clocks or watches are just instruments that let us feel locally that time is running or record time easier. We might perceive time as a way to measure the changes in the world - in other words, the sequence in which events occur. In distributed systems, defining the order of events through time is challenging due to the absence of a global clock that can be accessed without error. To avoid ambiguity, we call it global event ordering for the total ordering at the destination of events that occur in distributed systems on the Internet where the Network Time Protocol (NTP)[1] can be applicable. There may be slightly different global event ordering at different destinations due to different arrival times for the same event. All nodes are in the same destination if they refer to the same event ordering. An event is ready for ordering only once after existing events arrive and are ordered in advance. Therefore, for simplicity, late events are rejected, and the waiting or rejection threshold is determined by the destination. Once an event is ordered, it will not be changed by new events for total ordering and its transitivity. If the threshold is the maximum transmission time from the occurrence to the arrival, total ordering can be done at all destinations. Otherwise, total ordering can still be made only at each destination. We adopted NTP because it is popular and does not need special hardware support such as Precision Time Protocol (PTP) – IEEE 1588-2008.

Even with a local clock correction by NTP, machines run at slightly different rates, leading to time drift, which can accumulate to minutes per day without periodic adjustment. In fact, NTP servers can be polled by default with periods from 64 to 1024 seconds to dynamically correct the local timestamp according to traffic load. It can be configured by the client side at even slower rates. Furthermore, retrieving or correcting the time introduces indefinite delays, meaning that absolute total event ordering is unattainable, and only partial or causal ordering is possible, as proposed by Lamport in 1978[2] and later called the logical clock or Lamport timestamp. Although the logical clock supports partial order, it is not natural to what we feel on the wall clock and is not popularly implemented. By employing the notion of "happening before" and incorporating a bounded error of timestamps, clock synchronization in distributed systems becomes feasible. NTP provides this bounded error, maintaining time accuracy within tens of milliseconds on the public Internet and achieving submillisecond precision in local area networks under ideal conditions. Despite the lack of global event ordering, distributed systems can rely on centralized clocks or partial ordering for synchronization, as long as the resulting conflicts are acceptable. However, for events without causal relationship, such as concurrent events, timing conflicts might not be acceptable. For systems like blockchain or Internet of Things(IoT) with limited human interference or malicious error, achieving global event ordering with controllable bounded errors is crucial for trustworthiness and scalability.

Through NTP correction, the timestamp of the local clock provides a possible measurement to achieve deterministic global event ordering with bounded error. In other words, if the errors are acceptable, the order can be defined by timestamps. However, if the timestamps are not accurate enough to uniquely define the order, an additional unique measurement is required. We assume that events need to be ordered somewhere at the destination after they arrive. This means that the ordering might be made at some node

in the destination but used later at other places. However, events can be ordered by attributes before they arrive, such as occurrence or issue time. Therefore, the error is well defined at the time of occurrence and may need to be dynamically adjusted to reach transitivity for total ordering at the destination. Moreover, it is also necessary to ensure that the timestamp is trustworthy without tampering and that the design is feasible.

PowerTimestamp was coined with Estimable Proof-of-Work (EPoW)[3] to integrate the local timestamp and estimated computing power, as a unique indicator and priority, for global event ordering. The timestamp has an error, called TimeError, indicating that the actual time (UTC) of occurrence is in the range of the timestamp minus and plus multiple of TimeErrors for different percentage of confidence interval. The number of multiples of standard errors is specified in the block header with 2 bits. If the number is zero, an optional z score can be specified for the confidence interval as the fixed point number for the Bitcoin target. Events with a PowerTimestamp arriving at the destination form event groups and are ordered by event groups. An event group is formed with the events ordered by priority, whose timestamp ranges overlap with the main event. The main event of the event group is the earliest event with the earliest timestamp, the smallest TimeError, the highest priority, and the highest identity in order. By carefully defining the way an event group can be formed, global event ordering can be achieved.

Ordering is decided at the ready time after the events arrive; usually, the ready time is the same as the arrival time for convenience. If the ready time can be later so that the existing events have arrived, the global event ordering will be the same at all destinations. Since ordering by priority is to compensate for time errors, it needs to be limited to overkill. Setting the ready time properly after the arrival time might relax the overkill, which is out of the scope of this paper. The priority of the indicator might be converted so that it is not linear to the estimated computing power or even inverted for any considerations such as energy consumption, speculation, or business model for a certain kind of fairness. For example, if the lower estimated computing power has the higher priority, it might be fair with more inclusion, since the higher computing power already has advantages over the earlier timestamp. However, EPoW is not efficient enough to make PowerTimestamp feasible. Extended from EPoW, the General Proof of Work (GPoW)[4] is an efficient and reliable proof of work that could serve as a unique indicator of the estimated computing power. With GPoW, PowerTimestamp is trustworthy and feasible.

In this paper, we briefly describe related background in Section 2. Section 3 describes different types of PowerTimestamp design with optimized time or space in a prototype, and Section 4 discusses the implementation issues in the prototype. The paper is concluded in Section 5.

2. Background

Trust is an abstract concept and difficult to articulate. In order to design a trustworthy timestamp, it is crucial to first understand the nature of trust and how a PowerTimestamp is considered trustworthy. To facilitate this understanding, we first introduce blockchain technology, which is widely considered to be inherently trustworthy. In addition, we also briefly describe the GPoW consensus model as a means of establishing trust indicators. Then, we briefly introduce the Network Time Protocol.

2.1 Blockchain

In recent years, blockchain technology has attracted increasing attention from the public. Blockchain is often associated with key characteristics such as decentralization, immutability, transparency, and anonymity. The aspect that truly differentiates blockchain and makes it significant is its ability to function as a trust machine. Bitcoin was a pioneer in introducing the concept of decentralized currency by cryptography, the so-called cryptocurrency. The implementing technology was called blockchain few years after Bitcoin was coined. The key model that supports decentralized systems and statistically reaches consensus within a specific time frame is Proof-of-Work (PoW)[5]. The PoW model involves finding a random number, the so-called nonce, and embedding it within a block header as input to a hash function. If the generated hash value is lower than or equal to the target value set by the system, called valid, the individual who found the nonce is deemed to have completed a certain amount of work and might receive a reward, the so-called Proof-of-Work and mining.

By the SHA-256 hash function used in Bitcoin, it is very unlikely that one will determine in advance who will receive the next mining reward. Using a cryptographic hash function such as SHA-256, a small change in the input will greatly change the output. Therefore, the nonce in the block header need not be random to make the hash value random enough with little collision. This randomness introduced by SHA-256 ensures a level of fair competition within the Bitcoin network. Increasing computing power becomes a means to obtain greater rewards, since trying more nonces in each round of mining increases the likelihood of finding a valid nonce. The fairness of Bitcoin makes people think of it as a trustworthy system in the belief that the more work, the more reward. However, it still lacks clear metrics to quantify the level of trust it provides. There are many other consensus models, e.g. Proof of Stake(PoS). Despite the energy consumption problem, PoW is still the most reliable model for blockchain, with the largest market since lunching.

2.2 General Proof-of-Work (GPoW)

Based on PoW, EPoW, the US patent[6], records the individual two nonces of the highest and lowest hash values to estimate how many trials of nonces are in each round of mining as an indicator of the estimated computing power of the individual mining nodes. Extended from EPoW, GPoW, the ROC patent[7], is also estimable. The objective is to find the first m, the conservative GPoW model, or the best m, the aggressive GPoW model, valid nonces such that their hash values, individually, were lower than or equal to the target value no later than reaching the maximum number n, of nonce trials. The value of GPoW is the mean of the m hash values. The smaller GPoW indicates the higher estimated computing power. It is very unlikely that you have the same GPoW value. If it does happen, we can leave it to users to decide which one wins, usually by the object ID, the hash of block header, or just tie. Statistically, conservative and aggressive models are shown to be equivalent with the same probability density function (PDF), and so is the cumulative distribution function (CDF). Since the random variable of GPoW is in the beta distribution, the behavior of GPoW can be described in simple formulas of m and n to adjust the behavior of the system. Following the definition of PoW, a GPoW also stands for the information sent to receivers for verification.

The coefficient of variation (CV), defined by the ratio of standard deviation (square root of variance) to mean: $CV = \frac{\sqrt{\text{variance}}}{\text{mean}}$, can be used as a trust indicator $C = 1 - CV$. To make it simple, by adopting conservative GPoW, let the number of maximum trials n, be m, and the target value be $\frac{m}{m+1}$. For example, for m in [1, 2, 3, 5, 10, 1000], the trust indicator C corresponds to specific values: [33.3%, 57.1%, 69.2%, 80.6%, 90.0%, 99.9%]. That is to say, theoretically, trying 1000 nonces, the corresponding output value represented by the mean of hash values is trustworthy in the long run with a trust indicator of 99.9% or the coefficient of variation is 0.1%. If only 10 nonces are tried, it can reach 90%. In this paper, more than 90% or 1000 is considered high and less than 10% or 10 is considered low. That means GPoW is trustworthy, efficient, and feasible. When m is 1, GPoW behaves like Bitcoin, but the trust indicator is only 33.3%. Note that Bitcoin might try nonces for more than 10^{20} times in a round of mining to reach enough security. Given n and m in GPoW, we can always find a target value to satisfy some system need, such as the successful rate of building a GPoW or mining a block in a blockchain. However, the trust indicator would change accordingly and need a double check. If some configuration conflicts or security concerns arise, the use of a low target value to protect tampering might be replaced by an additional encryption mechanism on the GPoW or embedded in another protecting system such as blockchain. This is beyond the scope of this paper. With estimable computing power and flexible mathematic adjustment, GPoW can avoid increasing energy competition like PoW used by Bitcoin while maintaining an even higher level of security.

2.3 Network Time Protocol

Network Time Protocol (NTP) is a protocol designed to synchronize computer system clocks. Coordinated Universal Time (UTC) is based on International Atomic Time (TAI), which is a weighted average of hundreds of atomic clocks around the world. UTC is within about one second of mean solar time at 0° longitude, the currently used prime meridian, and is not adjusted for daylight saving time. The goal of NTP algorithms is to minimize both the time difference and the frequency difference between UTC and the system clock. When these differences have been reduced below nominal tolerances, the system clock is said to be synchronized with UTC. It operates within a client-server architecture, where time-stamped packets are exchanged between the client and the server to determine the time error or offset between the two machines. This offset information is then used to adjust the system clock on the client machine. To achieve reliable time synchronization, a client typically connects to three or more servers simultaneously. Using sophisticated algorithms, NTP evaluates and filters potential sources of misleading time information. This ensures that accurate time synchronization is maintained even if one or more servers become unavailable or even provide incorrect data. The error of timestamps in a distributed system can be robustly estimated in a given confidence interval[8], even if there are outliers from malicious attackers. With Network Time Protocol (NTP)[9], it can usually maintain the time error to within tens of milliseconds over the public Internet, and it can achieve a better accuracy than a millisecond in local area networks under ideal conditions. In addition, asymmetric routes and network congestion can cause errors of 100 ms or more[10].

In RFC 5905 (NTP4 Specification), time-related factors such as timestamp, offset,

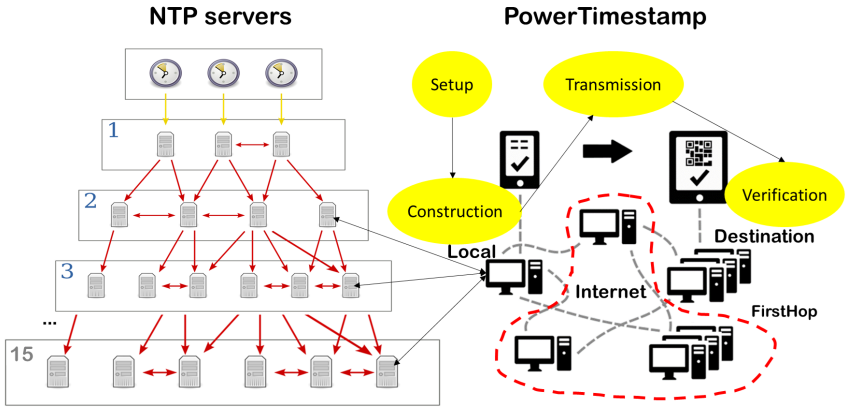

Fig. 1. The system architecture of NTP and PowerTimestamp.

delay (round trip), jitter (an estimator of offset), wander (oscillator frequency stability), etc. are clearly specified with simple formulas. Since timestamps are synchronized or corrected in an open network with open frequencies, the protocol and formulas help to bound the errors but still cannot support total ordering. The error is bounded because the servers are hierarchically connected with limited access, usually by mainly local machines. The event closer to the servers thus has more accurate timestamps. Using the capabilities of NTP, PowerTimestamp seals the events locally and can determine the reliable total ordering of events at the destination, ignoring errors in between as long as they arrive and are accepted within delay.

3. PowerTimestamp

In this section, we will describe the design and implementation of PowerTimestamp. m still represents the number of valid nonces selected in the GPoW, and n represents the maximum number of trials in which users attempt to find the nonces in a GPoW. If mn in the PowerTimestamp block header is 0, m and n are defined locally by users; otherwise, they are defined by participants. As shown in Fig. 1, PowerTimestamp takes advantage of NTP where local computers connect to NTP servers to correct the local clock. Starting from the first stratum connected to accurate time sources (the atomic clock), NTP servers can stack up to 15 strata. Depending on the stratum on which the server resides and how close the local machine is connected to the server, the timestamp synchronized might have differences ranging from a few microseconds to hundreds of milliseconds. On the Internet without traffic jam, usually, the round-trip latency is around tens of milliseconds. The FirstHop stands for the potential first hops of computers that support PowerTimestamp on the way to the destination, similar to the eight outbound neighbors collected for broadcasting on the Bitcoin network. The round-trip latency is measured and maintained between the FirstHop and local machines. At FirstHop, the PowerTimestamp will be verified and rejected to be sent further to the destination if it is a potential faking in time. Before using PowerTimestamp, users need to set up the requirement of the trust indicator,

the time for the last trial, the size of PowerTimestamp, and the security preference, etc. The requirement is usually requested from the destinations or negotiated in advance by all participants.

Using GPoW, as PoW by definition, a PowerTimestamp should take much more time in construction than in verification. However, if m is large, it might take too much network bandwidth and transmission time. Therefore, there are different types of PowerTimestamp with optimized time or space. All types can be compared with each other in the ordering of events if the participants agree regardless of the time or space issue. There might be different versions of PowerTimestamp, and only the same version can be compared, with the same GPoW model and priority scheme. The configuration will be fixed prior to the construction stage. Some requirements may not be met because the system is changing dynamically. However, the determinism of event ordering should be kept in all circumstances. After the construction stage with the best effort, the PowerTimestamp can be sealed in a GPoW. Then, the PowerTimestamp can be transmitted to any destination simultaneously or repeatedly one after another for verification and ordering.

3.1 Design

To construct a reliable timestamp, a straightforward design for PowerTimestamp can integrate the physical timestamp obtained from the computer into the GPoW with a set of m nonces. Most computers today have a high-resolution clock of nanoseconds. Although NTP can be with a precision of hundreds of picoseconds or higher[10], different from the 32-bit Unix-type timestamp in Bitcoin, PowerTimestamp has a 64-bit unsigned timestamp in nanoseconds following the standard C++ library. The TimeError takes an unsigned one-byte integer in milliseconds, and the maximum value is 255. If TimeError is 0, we can directly compare the timestamp, and the unique indicator is ignored. Following the same data type used in Bitcoin, the data type of nonces in GPoW is normally also a 32-bit unsigned integer. The nonce is tried starting with 0 incremented by 1. If a nonce is found, the next nonce starts on the nonce plus 1. For space concern, one type of PowerTimestamp can be any size s from 1 to 31, with a size of $s \times i$ bits for the ith nonce, Nonce_i. The Nonce_m contains all previous nonces. The next nonce is tried starting with 0 of the s bits attached to the least significant end of the previous nonce, still incremented by 1. For this type to find a successful GPoW, backtracking of iterating previous nonces might be needed. If no nonce can be found even with backtracking, we can update the timestamp to try all over again. We can increase the target value to decrease the time it takes to construct a PowerTimestamp. A PowerTimestamp should be constructed in a bounded time after its timestamp. The size field of version in the block header is set to 0, if the PowerTimestamp accepts any number of MTU (maximum transmission unit, 1500 bytes in Ethernet). The size can be up to 15 MTUs.

For the different types of PowerTimestamp, the timestamp can be verified by any of the nonces transmitted. However, if the target value is not set low enough, it might be too easy to reconstruct one, i.e. it is not secure. Usually, the successful rate of a GPoW is set to high to allow PowerTimestamp to be constructed easily in time. If m is not too high, it is still possible to find a target value for security purposes. Otherwise, the trust indicator or m should be decreased. If security is a concern, the PowerTimestamp must be signed with the Public Key Infrastructure before transmission. After verification at each destination, the PowerTimestamp information can be retrieved, if necessary, for a secure

and intact comparison. We can ensure that the timestamp is in the moment, close to UTC, when the PowerTimestamp is constructed in a bounded error of time.

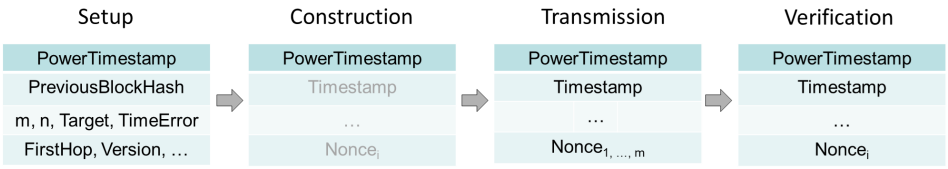

Fig. 2. The stages of simple PowerTimestamp.

3.1.1 Simple PowerTimestamp

After setup with configured parameters, as shown in Fig. 2, the timestamp and each Nonce_i can be tried in the construction stage. Then, put together in the PowerTimestamp block header to transmit to destination for verification on each nonce. The GPoW value can be calculated accordingly for comparison. We call it a simple PowerTimestamp.

3.1.2 Incremental PowerTimestamp

However, if m is large, the m nonces might take too much network bandwidth and transmission time, especially when it needs to be broadcast in applications such as blockchain. When constructing the nonces, we can keep the previous nonces in order and attach the ith nonce at the end, as $\text{Nonce}_{1,\ldots,i}$. The nonces should be verified later in the same way. Unlike the simple PowerTimestamp, the order of nonces matters. We call it incremental PowerTimestamp. In this way, each nonce need not to be 32 bits. Actually, depending on the target value, one bit is also feasible. The size of nonces can thus be reduced to one thirty-second. However, there might be backtracking to find all the valid nonces. It might be time-consuming. For a PoW-like model, it is usually acceptable to take more time building for less space requirement.

3.1.3 Thin PowerTimestamp

The smaller the size of the nonces, the higher the target value needed for a successful GPoW. The higher the target value, the less secure the system. Is there a way to take care of both size and security? Although in GPoW, the normalized target value 1 is added for easier analysis of beta distribution, we can ignore it in practice for security. That is, some trials might not be valid and easier error checking might be performed. Therefore, following the same monotonically incremental generation of nonce for a successful GPoW, as long as the timestamp is fixed, the m valid nonces are unique. If m is large, it is difficult to guess in advance the value of the last nonce, Nonce_m. We can put only the last nonce, Nonce_m, to be transmitted for verification. If verification time is not a concern, we can reconstruct all the nonces on the destination side and verify by whether the last nonce is the same. We call it a thin PowerTimestamp.

3.1.4 Quick PowerTimestamp

Once the timestamps for different PowerTimestamps can be verified and ordered, the following reconstruction or verification can be skipped, and the GPoW value can

be calculated later on demand. Similarly to the thin PowerTimestamp, we can add the nonces, $\text{Nonce}_{(1)}$ and $\text{Nonce}_{(m)}$, for the smallest and the largest hash values, respectively, to transmit for verifying timestamp first, and then form a range of GPoW value. If the range cannot still prioritize the PowerTimestamps, we then continue to reconstruct and find the GPoW value for ordering. We call it a quick PowerTimestamp.

3.1.5 Compressed PowerTimestamp

If m is too large so that the PowerTimestamp cannot fit the size specified in the block header. The nonces of simple and incremental PowerTimestamps need to be lossless compressed before transmission and decompressed for verification afterwards. The nonces are m increasing unsigned integers of the same size from 1 to 32 bits. The lossless increasing integer compression[11] has been well studied. The compression rate is data dependent and varies greatly, even greater than 1. The compression algorithm with the best compression rate is tried and specified in the version field of a byte, but the byte needs to be excluded from construction and verification for the nonces. If no algorithm can fit to the specified size, this would be Algorithm 0 and the last nonce would be used for the compressing result and processed as the thin PowerTimestamp. The rest of space can be filled with other evenly selected nonces starting from Nonce_1 for more security checks. This would be Algorithm 1. The different algorithms implemented will be numbered and updated on the to-be-released PowerTimestamp Github site. We call it a compressed PowerTimestamp.

3.2 GPoW Issues

Since there is no mechanism like Bitcoin to reach consensus on GPoW in PowerTimestamp, PowerTimestamp randomly chooses a node in FirstHop in the middle[8] of round-trip latency to transmit for better security. We tested on a 64-bit Ubuntu 22.04.5 LTS with 16 GB RAM and an Intel® Core™ i7-8750H CPU, and 64 GB RAM and an Intel® Core™ i9-12900F CPU, to construct successful thin PowerTimestamps. Both the i7 and i9 machines behave similarly, except that i9 is faster for large cases, while i7 is faster for small cases since the CPU clock rate of i9 changes dynamically, thus with much more variance in time, but better performance on average.

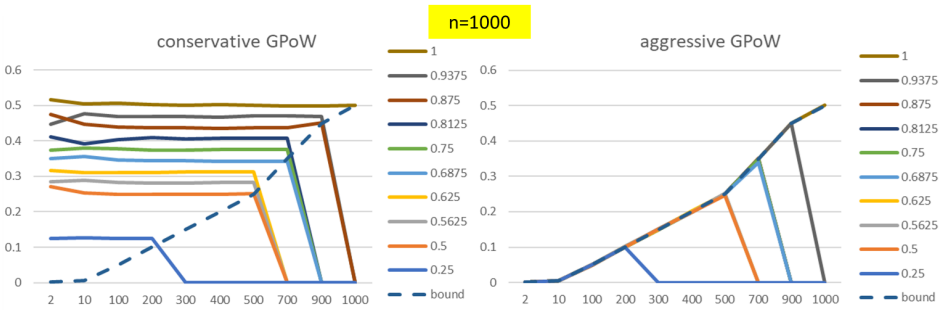

Fig. 3. The GPoW mean for target 0.25 to 1.

Although the two GPoW models are statistically equivalent, they behave differently in practice. On average for 100 rounds on the i7 machine, as shown in Fig. 3, the X axis

is m, n 1000, the Y axis is the mean GPoW, each line represents using different target values from 2.5 to 1, and the dashed line is the theoretical bound. The conservative model behaves consistently for each target value after about $m > 100$, but far from the theoretical bound. The aggressive model fits the theoretical bound perfectly, but looks as if there is no difference in the different target values. However, since the hash values are very fine with 256 bits, they are still differentiable. This is because the hash function SHA-256 is not random enough or is too evenly distributed, and is avoiding collision for hashing. For a "perfect random" hash function, the CV for conservative GPoW would have more slope, and aggressive GPoW would have more difference. Note that, as m increases, with the lowest target value first, there may not be a successful GPoW and the mean drops to zero. No matter how m and n change, the mean value changes linearly and is bounded at 0.5. As shown in Fig. 4, CV also performs differently when m is low. However, as m grows, the CV is low and close to the theoretical bound, and stable enough for a trust indicator, as is the mean for comparison. Note that the scale is not evenly distributed.

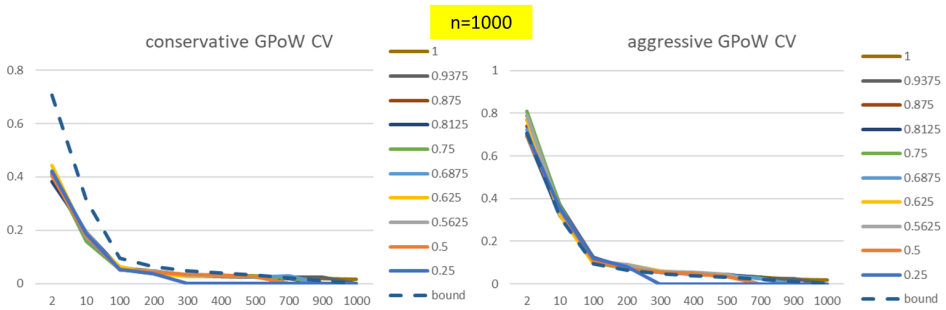

Fig. 4. The GPoW CV for target 0.25 to 1.

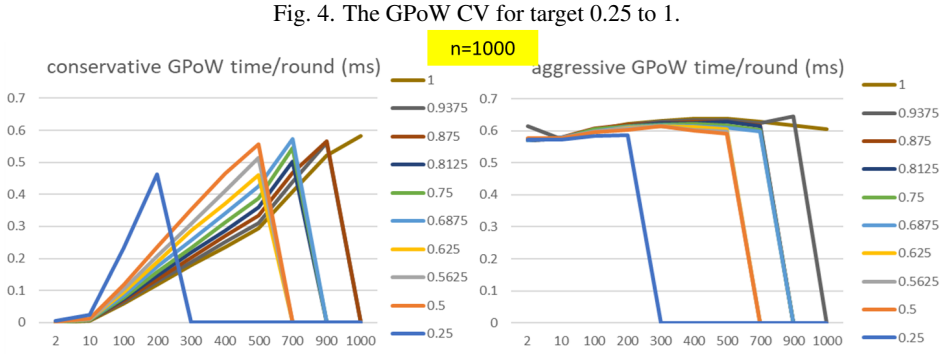

Fig. 5. The GPoW constructing time for target 0.25 to 1.

As shown in Fig. 5, the conservative GPoW constructing time per round grows linearly with m starting from a few microseconds. The aggressive GPoW constructing time per round starts around 0.6 milliseconds, grows slowly, and drops a little because it is more difficult to find a better nonce after some nonces are found; the lower the target value, the more difficult. As shown in Fig. 6, although there might not be a tight theoretical bound for execution time and its trust indicator, CV is lower on average as m or n grows. However, the variance is sensitive to the load of the system, especially when the CPU clock rate can be boosted on demand. The aggressive model might even incur

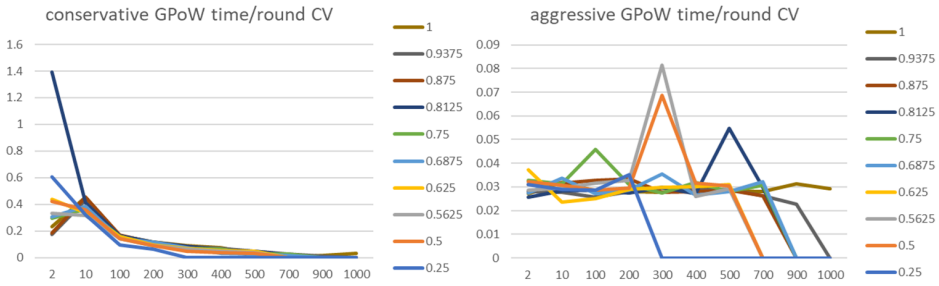

Fig. 6. The GPoW constructing time CV for target 0.25 to 1.

more time for backtracking in constructing incremental PowerTimestamp because the latest nonce might replace a previous one and these nonces are dependent, causing cascading replacement and large variance. Therefore, incremental PowerTimestamp is only suitable for the aggressive model when m is small. Since SHA-256 is a trusted and popular standard for its good features, such as low collision and high determinism, optimizing for randomness might not preserve other goodness. We would better not modify it to fit the theoretical bound or lower the variance. How to generate nonces for more random hash values without losing good features might be the only way out and is our future work.

3.3 Global Event Ordering

With the help of the synchronized NTP timestamp and GPoW as a unique indicator, PowerTimestamp can support global event ordering. The question is how accurate it can be. If faking is a concern, we cannot only count on local machines and FirstHop is the best we can rely on by randomly choosing one from the FirstHop pool to verify for further transmission. Since the information of PowerTimestamp is sealed in GPoW, the problem left now is how to detect potential faking in local machines. After setup, all the PowerTimestamp configuration in block header is fixed, except those for compression. Once all valid nonces are found, the PowerTimestamp is sent to FirstHop right away; otherwise, it is also a potential faking. If we can estimate the duration between setting the timestamp of PowerTimestamp and arriving with the timestamp at the FirstHop, we might be able to detect the faking or we limit the faking in the TimeError range.

The duration before transmission includes 4 elements of constant routines, hashing for each trial, maintenance of the first m valid nonces, and insertion of the better nonce sometimes to replace a previous valid nonce for aggressive GPoW. Suppose that the unknown variables for the part of average time in the constant work for each element are X, Y, Z, and P, respectively. For possible nonce compression, we exclude and assume that the time spent can also be estimated at FirstHop. For s successful PowerTimestamps with target t, the duration $D_{m,n,t}$ is

$$\begin{cases} (X+Ym+Zn')s & \text{// for conservative GPoW.} \\ (X+Ym+Zn+P(tn-m)F(m)\log m)s & \text{// for aggressive GPoW.} \end{cases}$$

, where n' is how many trials are done, including valid and invalid ones, and $F(m) = \sum_{i=m+1}^{n}(\binom{n}{i}(1-\binom{i-1}{m}/\binom{i}{m}))(t)^i(1-t)^{n-i} = \sum_{i=m+1}^{n}\frac{m}{i}\binom{n}{i}(t)^i(1-t)^{n-i}$ is the CDF of the

better nonces after m valid nonces are found. Because for cases with $i > m$ valid nonces, only the latest one, if not the largest, and the previous $m-1$ ones are kept. We use 3 simultaneous linear equations with different m to solve the 3 unknowns for conservative GPoW. Unfortunately, few can be solved, let alone the aggressive GPoW. The solved ones still vary a lot because the time for constant work varies due to system load, interrupt, etc. More research and improvement can be done to approximate solutions by range arithmetic, but this is beyond the scope of this paper. However, we believe that the execution time and error for a fixed amount of work should be bounded for any machine.

Suppose that there is a trusted database (or blockchain) supported by manufacturers or peers with the mean and variance of constructing time for different configuration of PowerTimestamps on different machines, the estimation becomes easy by lookup or interpolation. Moreover, in the setup stage, several PowerTimestamps with the same configuration can be precomputed continually and linked in the PreviousBlockHash field of the block header in preparation and sent separately at the same time to FirstHop for estimating the construction time and transmission time by execution again or analysis, even if FirstHop is on a different kind of machine. The information should be inserted with care, e.g. within certain errors, to ensure accuracy in the database for future references. In the long run of maintenance, the database should be able to provide fewer and fewer errors in time until bounded. With this faking protection mechanism, the TimeError can be determined and global event ordering can be possible in the accuracy of TimeError if the following event groups can be well defined for total ordering.

3.4 Event Groups

We do not have a global clock for all events with a definite time of occurrence to form a total ordering such that all events in the system occur in the same order on all nodes. Total ordering has no problem of transitivity by the imaginary global clock at any node. In reality, we assume that the events need to arrive at some destination to do ordering for comparison. We cannot support a comparison in the sky when the events have not yet arrived. Therefore, it depends on the arrival time. That is, the ordering might be different at different destinations unless we wait for existing events to arrive. Moreover, for any mechanism to replace the global clock, there might be some adjustment in ordering. The adjustment needs to follow the transitivity law for total ordering as well. However, in reality, we might not always reject late events. Nevertheless, events might only need to be compared in a few shots at one destination, and no ordering needs to be kept for good. Therefore, the design of PowerTimestamp also needs to balance cost and performance.

A PowerTimestamp is embedded in an event to form a global event ordering such that the events are ordered only after they arrive at some destination. If the order needs to be all the same at different destinations, it can be done after existing events arrive. In fact, we only need to wait for half of the longest round-trip delay, which is less than 1 second conservatively, or just reject the event late for more than 1 second. Otherwise, we need to reach a consensus such as blockchain for the arrival time at different destinations, which is possible with more delay, but that is outside the scope of this paper. Now the problem left is following the transitivity law. If the law can be followed, that means new events will not change existing ordering, late events can also be accepted for total ordering at one destination, or the late event is the first arrived with consensus at all destinations.

Events with a PowerTimestamp arriving at a destination form event groups and are

ordered by event groups. An event group is formed with the events ordered by priority if the main event and the others are overlapped, i.e. the TimeError range (timestamp-TimeError, timestamp+TimeError) overlaps, excluding the ends. The main event of the event group is the event with the earliest timestamp, the smallest TimeError, the highest priority, and the highest identity in order. Using the earliest timestamp and the highest priority is to follow the transitivity. Using the smallest TimeError is to reduce the size of a group as much as possible for higher accuracy. The highest identity is the lowest hash value of events with the same priority. The event groups are formed by the events in the same order as well. Therefore, the main event might not have the highest priority in the group. We assume that no hash value will be the same for different events. An event is also an event group itself. The event groups are ordered by the main event. An event is earlier if its event group is earlier, with higher priority, or with the same priority and higher identity in a group. Therefore, it is a strict total ordering.

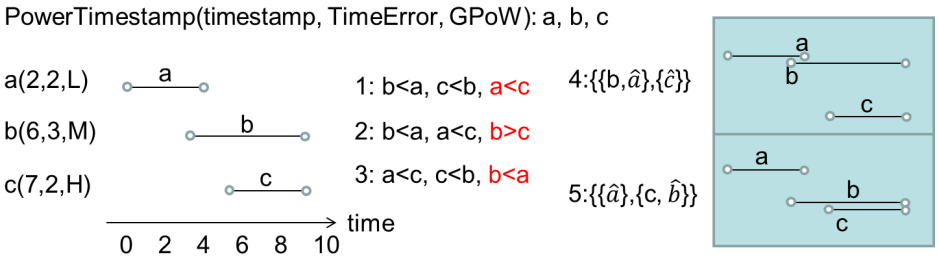

Fig. 7. Event groups.

As shown in Fig. 7, a, b, c are PowerTimestamps with timestamp, TimeError, and GPoW in a 3-tuple such as (2,2,L), (6,3,M) and (7,2,H), respectively, for some events, where H, M and L stand for high, medium and low priority as a GPoW, the smaller value, the higher priority. Note that the ends are not included. In the basic definition of PowerTimestamp, PowerTimestamp x is earlier if the timestamp plus the sum of both TimeErrors is smaller than the other timestamp of PowerTimestamp y, that is, the timestamp is smaller and the range is not overlapped, or the priority is higher, denoted as $x < y$. For PowerTimestamps a, b, c, the relationships of each two are $b < a$, $c < b$ and $a < c$. Any of the two relationships violate the transitivity law as in cases 1, 2, 3 in the figure. If we form event groups, shown in braces in order and the main event is hatted, and compare by event groups as in case 4, the transitivity problem can be solved. However, if an earlier PowerTimestamp a arrives later than b,c is formed as in case 5, there would be different orderings and the total ordering might be violated. Therefore, we need to wait for existing events to arrive, but not future irrelevant events, and form event groups with the earliest unformed event first. An event only needs to consider joining the previous event group or forming a new one if there are no late events. If late events are allowed, the closest event group is chosen to join before forming a new one if no existing events are changed in the group because the main event is replaced. The distance from an event group follows the same order as above for the main event, except that the earliest timestamp is replaced by the shortest distance of timestamps. Then, the total ordering can be done in global event ordering. Now, we are ready to determine the TimeError in the next section.

4. Implementation Issues

After theoretical design and prototype development above for PowerTimestamp, there are still issues to be addressed for real implementation.

4.1 Trust Indicator

The theoretical trust indicator (TI) bound introduced by GPoW is simplified to allow easy understanding of the system and initializing the configuration. The real performance will be different, especially since we found that SHA-256 is not random enough. By 100-round average, as shown in Table 1, for a target value greater than 0.25 and $n = 1,000$, conservative GPoW TI is higher than aggressive GPoW in general, and they are still close to the theoretical bound after $m > 500$, let alone the randomness of SHA-256. No matter the target value, the trust indicators of conservative GPoW and aggressive GPoW can reach greater than 95% for m is greater than 200 and 500, respectively. Even in $m = 10$, the conservative GPoW TI still has 80.6%. For $n = 100$, the trust indicators are lower than $n = 1000$ with the same trend in terms of the same $\frac{n}{m}$, none greater than 95%, and the aggressive GPoW is even lower. However, if time is an issue, the trust indicator is still not too bad with a few microseconds per GPoW. Changing n to 10,000 and 100,000, respectively, the trend is similar and the indicators are more accurate in terms of the same $\frac{n}{m}$ with slight variation. Both GPoW models can reach higher than 95% for m higher than 200 and 700, respectively. To reach the high-trust indicator, m must be close to n. The larger n, the higher the trust indicator. However, it also takes roughly 10 or 100 times that time when $n = 1,000$. If time is not a concern, reaching 99.9% is possible for $n \geq 100,000$. The actual building time is the most significant in using PowerTimestamp.

Table 1. Lowest TI (%) for $n = 1,000$, 100-round average for target 0.25 to 1.

m	10	100	200	400	500	700	900	1,000
Conservative GPoW	80.6	93.6	95.2	96.9	97.2	97.2	97.7	98.3
Aggressive GPoW	62.8	87.7	90.8	94.7	95.5	96.9	97.5	98.2
Theoretical bound	68.6	90.5	93.7	96.1	96.8	97.9	98.9	99.9

4.2 TimeError

The TimeError is the key concept defined in the PowerTimestamp that represents the time error of any two events based on the PowerTimestamp with respect to NTP servers. Actually, it is the time error that sets the timestamp when constructing the corresponding GPOW since the timestamp is sealed afterward before verification. Suppose that the transmission, propagation, queueing, and processing time errors to transfer the same type of event packets with the same type of PowerTimestamp to FirstHop can be ignored, or, with the construction time error, these errors can be detected and controlled by FirstHop within certain percentage of the construction time. The TimeError can be set by experience, such as 10% of the construction time plus the NTP time error on the local machine, since the trust indicator can easily be higher than 90%. We assume that the NTP time

Table 2. The GPoW constructing time(ms) and its TI(%) for target 0.5 to 1.

n=100, m		2	10	50	90
Conservative GPoW	Time	0.003→0.002	0.014→0.006	0.064→0.030	0.059∼0.052
	TI	-56.8∼83.2	54.1∼91.1	79.1∼96.3	84.6∼99.3
Aggressive GPoW	Time	0.068∼0.058	0.070∼0.060	0.071∼0.060	0.063∼0.061
	TI	88.6∼98.9	81.2∼98.5	86.8∼98.2	94.8∼97.2
n=1,000, m		10	100	500	900
Conservative GPoW	Time	0.012→0.006	0.119→0.059	0.557→0.293	0.565→0.521
	TI	54.5∼68.3	83.0∼85.8	95.0∼98.3	98.5∼99.0
Aggressive GPoW	Time	0.580∼0.573	0.607∼0.595	0.637∼0.592	0.646∼0.616
	TI	96.6∼97.6	95.4∼97.5	94.5∼97.2	96.9∼97.7
n=10,000, m		100	1,000	5,000	9,000
Conservative GPoW	Time	0.119→0.059	1.143→0.586	5.867→2.898	5.587→5.254
	TI	73.7∼87.0	96.6∼98.8	98.1∼99.0	98.9∼99.0
Aggressive GPoW	Time	5.737∼5.709	6.401∼5.937	6.603∼6.064	6.467∼6.365
	TI	98.3∼99.1	98.2∼98.9	98.0∼99.4	98.6∼98.7
n=100,000, m		100	1,000	10,000	90,000
Conservative GPoW	Time	0.119→0.059	1.153→0.579	11.68→5.871	57.08→53.57
	TI	79.1∼95.1	96.9∼99.1	98.1∼99.2	99.3∼99.6
Aggressive GPoW	Time	57.37∼57.16	58.29∼57.78	62.97∼61.06	66.17∼64.37
	TI	99.4∼99.7	99.4∼99.7	99.3∼99.7	99.4∼99.4

→:decreasing, ∼: oscillating.

error of FirstHop is the same as that of the local machine, or that the error is negligible. As shown in Table 2, we measure the construction time of the GPoW in a range for target values of 0.5 to 1. The conservative GPoW time decreases monotonically in the range and that of aggressive GPoW oscillates with slight variation, as shown in Fig. 5. The variation for target values less than 0.5 might be too large and excluded to be a trust indicator. With $m = 2$, it could be negative and untrustworthy. Therefore, we suggest using those greater than or equal to 0.5. However, if we know the event order is not a concern and there is a grace period longer than the TimeError summation of both sides, using small n and m to construct the PowerTimestamp faster would be a good policy to save computing power. Or just choose the type-0 PowerTimestamp, ignoring GPoW.

As shown in Table 3, we measure the time error in the 1,000-round average in WAN and LAN through NTP 4. The positive value means that our time is faster. Since PowerTimestamp is used more at few shots than in a long period of time, we used the original mean and standard deviation (SD) instead of taking the absolute values. WAN time errors are collected from regions supported in the NTP pool, tw.pool.ntp.org. The NTP pool is a dynamic collection of networked computers that volunteer to provide highly accurate time via the NTP to clients worldwide. The NTP pool automatically distributes NTP servers from the pool, so the NTP servers obtained in the same region may vary each time. In the LAN experiment, we had five computers (node0 to node4) in our lab with 100 Mb Ethernet. We set node0 as the NTP server and the other four machines synchronized their time based on node0. The results did not show a significant offset from node1 to node2, node3,

Table 3. WAN and LAN time errors(ms).

Region\WAN offset	Mean	Mean(abs)	SD	SD(abs)
Global	-3.387	4.787	23.544	23.300
Taiwan	-2.320	2.966	29.373	29.314
Asia	8.082	15.737	41.321	39.052
North America	-0.859	6.958	34.788	34.095
Europe	7.337	9.761	19.161	18.046

LAN offset	node 2	node 3	node 4
node 1	-0.138	0.133	0.167

and node4 due to the stability of the LAN. From the results, it can be concluded that the TimeError can be determined deterministically in tens of milliseconds. 100-millisecond TimeError would be reasonable for most cases in the 95% or higher confidence interval where the verification can be achieved by multiple TimeErrors, such as the z score 2.58 for 99%. If NTP servers, local machines, and FirstHop can be in a LAN, TimeError can be submillisecond for some n, m, t in combination. The tables can be stored in public and updated periodically for lookup at destinations to decide the type of PowerTimestamp for local machines to follow for comparison. Suppose that the NTP servers do not change frequently; the errors can stabilize soon after all. We suggest using the 1024 polling interval in the local machine to reduce jitter and save energy.

4.3 Priority Scheme and Rewarding

Bitcoin uses a constant block reward to encourage miners to maintain its operation. The PoW mechanism results in computing power competition because the more computing power, the more rewarding it is linearly. In addition, reward in general includes processing and results so that linearity might not reflect the reality. Ideally, the estimability of EPoW and GPoW for computing power helps to award non-linearly and reasonably, but it is still possible to be abused by monopoly of computing power. Especially for PowerTimestamp, it is more used in few shots than in a long period of time. High computing power would make the result easier to manipulate. Although increasing n would benefit users with high computing power, decreasing n would also lower the trust indicator and benefit users with low computing power. It depends on how people value computing power and trust to participate, maintain, and operate the whole system. Global event ordering provides a mechanism to sort things out, but it might not be necessary to bound with rewarding. The ordering and rewarding can be decoupled. Inverting the priority scheme of GPoW to make it greater value with higher priority might relax the problem, but it might also discourage participation. We can even make one more hash on GPoW on rewarding to make it irrelevant to ordering. Moreover, the priority schemes can even be mixed with each other to fit the real world. Priority scheme 0 in the block header represents the largest value with the highest priority by comparing the priority value directly. Priority scheme 1 represents the lowest value with the highest priority by comparing the negated priority value. Priority scheme 2 represents a new irrelevant order by compar-

ing the one-more-hashed priority value. Priority scheme 3 represents a mixed scheme by comparing the weighted sum of other scheme values defined by the system in charge. The system can test and adjust to find the best weights using the GPoW mathematical formulas and the time-driven design. Although the basic order is fixed, the priority scheme can be adjusted periodically to fit other requirements, such as social needs.

5. CONCLUDING REMARKS

We propose a detailed PowerTimestamp design by NTP 4.0 and GPoW with a prototype to support global event ordering and discuss implementation issues with solutions. Global event ordering means that if we can wait for existing events to arrive or reject late events at all destinations, total ordering can be performed with a time error of hundreds of microseconds to hundreds of milliseconds for a confidence interval or trust indicator higher than 95%. If there are late events to be accepted, the total ordering can still be done at one destination or consensus for arrival times with more delay for all destinations. If the network round-trip delay is less than 500 milliseconds, the total ordering of 95% events can be done by waiting a second after any event arrives with a time error or faking in 100 milliseconds. Although accuracy can be improved by improving the randomness of the hashing function and a sophisticated tool is needed to friendly configure the system to use PowerTimestamp and maintain FirstHop for security, PowerTimestamp is the first solution for total ordering. This is a breakthrough in distributed systems that allow synchronization to become decentralized, deterministic, and feasible by the occurrence of events. We will continue on our future work and open source as soon as possible for any possible cooperation and realization.

ACKNOWLEDGMENT

This research was funded in part by the National Science and Technology Council, Taiwan, under the grant NSC 111-2221-E-002-124-MY2 and NSTC 113-2634-F-002-001-MBK.

REFERENCES

1. D. L. Mills, “Internet time synchronization: the network time protocol,” *IEEE Transactions on Communications*, Vol. 39, no. 10, 1991, pp. 1482—1493.
2. L. Lamport, “Time, clocks, and the ordering of events in a distributed system,” *Communications of the ACM*, Vol. 21, no. 7, 1978, pp. 558—565.
3. C.-W. Hsueh and C.-T. Chin, “Epow: Solving blockchain problems economically,” in *Proc. of the 14th Conference of Advanced and Trusted Computing*, 2017, pp. 1–8.
4. C.-W. Hsueh and C.-T. Chin, “Toward Trusted IoT by General Proof-of-Work,” *Sensors*, Vol. 23, no. 1, 2022, p. 15.
5. S. Nakamoto. (2008) Bitcoin: A Peer-to-Peer Electronic Cash System. [Online]. Available: SSRN:http://dx.doi.org/10.2139/ssrn.3440802

6. C.-W. Hsueh, *Estimable Proof-of-Work for Blockchain*. Patent No. US 10,965,466 B2, March 30, 2021.
7. C.-W. Hsueh, *METHOD AND COMPUTING DEVICE FOR PROOF-OF-WORK RELATED TO BLOCK MINING*. Patent No. R.O.C. I841910, May 11, 2024.
8. Y. Cheng, *Algorithmic Solutions and Applications for Robust Estimating with Scale-contaminated Normal Error Function*. Master's Thesis, Department of Computer Science and Information Engineering, National Taiwan University, July 2018.
9. D. Mills, "Improved algorithms for synchronizing computer network clocks," *IEEE/ACM Transactions on Networking*, Vol. 3, no. 3, 1995, pp. 245–254.
10. Wikipedia. (1999) Network Time Protocol. [Online]. Available: https://en.wikipedia.org/wiki/Network_Time_Protocol
11. C. S., L. D., K. O., and G. R., "Better bitmap performance with Roaring bitmaps," *Software: Practice and Experience*, Vol. 46, no. 5, 2015, pp. 709—719.

Author one Chih-Wen (Steven) Hsueh is an associate professor at the Graduate Institute of Networking and Multimedia and Department of Computer Science and Information Engineering at the National Taiwan University (NTUCSIE), R.O.C., since 2013, August, joined as an assistant professor in 2006 March. His research interests include real-time systems, distributed systems, operating systems, and blockchain. He received his MS degree in Computer Science from the University of Southern California in 1994, June, and his Ph.D. in Information and Computer Science from the University of California at Irvine in 1997, December. He joined as an assistant professor at the Department of Computer Science and Information Engineering at the National Chung Cheng University, R.O.C., in 1999, August, and an associate research engineer at ASIIS, in 2005, August.

Author two CHEN, YEN-SHUO is currently a Ph.D. student at the Graduate Institute of Networking and Multimedia, National Taiwan University. He graduated with a Master and Bachelor degree from the Department of Biomechatronics Engineering at the same university in August 2022 and June 2020. His current research focuses on distributed systems and blockchain technologies.