Lamport Spacetime Diagrams and Logical Clocks*

Grygoriy Zholtkevych[1,*,†] and Oleksandr Berezovskyi[1,*,†]

[1]Ivan Franko National University of Lviv, Universytetska Str 1, 79001 Lviv, Ukraine

Abstract

This paper addresses the critical challenge of event ordering and causality determination in asynchronous distributed systems, a problem of growing importance with the widespread adoption of Distributed Ledger Technology (DLT). The authors highlight the insufficiency of physical time for ordering transactions due to clock skew and the inability of physical timestamps to reflect cause-and-effect relationships. To provide a rigorous analysis, the paper presents a formal mathematical model for Lamport spacetime diagrams, which visualize execution scenarios by mapping events into logical spacetime. The model defines a set of events (local actions, message sending, and message receiving) and employs a "trigger" function to model message transmission between nodes. Central to this framework is the "happen before" relation, which serves as a formal model for causality in distributed computations. The authors demonstrate that the existence of a logical clock is a necessary and sufficient condition for the irreflexivity of the "happen before" relation, which is essential to guarantee the absence of system anomalies such as deadlocks. While the classic Lamport clock successfully orders events and identifies causal relationships, the paper identifies its primary limitation: non-injectivity. In concurrent systems, different events can receive the same timestamp, preventing the logical clock value from serving as a unique event identifier. To resolve this, the research examines a modified injective logical clock algorithm. This modification operates on a specific time scale by combining the standard logical clock with node identifiers. The authors prove that this modified function is an injective logical clock, ensuring that every distinct event receives a unique timestamp. This enhancement enables logical time to serve as a reliable and unique identifier, significantly improving data consistency, integrity, and functionality within decentralized DLT systems.

Keywords

distributed ledger technology, Lamport spacetime diagram, logical clock, “happen before” relation, causality, injectivity

1. Introduction

Nowadays, distributed storage and processing of information have become mainstream due to the growth of data and the spread of cloud technologies.

Centralized cloud services managed by large corporations offer convenient and scalable solutions.

However, this model cannot be considered as entirely trustworthy, particularly when it comes to data registries that must be accessible and reliable for the general public.

The key feature of that approach is that it requires a trusted third party responsible for the integrity, authenticity, and availability of the data.

Reliance on a single trusted center creates several risks that have been thoroughly studied in the scientific literature.

First, such a center becomes a single point of failure, making the system vulnerable to technical failures, cyberattacks, or unauthorized access [1].

Second, it creates an asymmetry of power where a trusted intermediary can censor, manipulate, or block access to data, which is contrary to the principles of openness and transparency [2].

*CMIS’26: The Ninth International Workshop on Computer Modeling and Intelligent Systems, May 5, 2026, Zaporizhzhia, Ukraine

[1,*] Corresponding author.

[†] These authors contributed equally.

grygoriy.zholtkevych@lnu.edu.ua (G. Zholtkevych); oleksandr.berezovskyi@lnu.edu.ua (O. Berezovskyi)

0000-0002-7515-2143 (G. Zholtkevych); 0009-0003-2241-3324 (O. Berezovskyi)

These problems are particularly relevant for registries that contain critical information, such as identity data, medical records, or patents.

An alternative that offers a fundamentally new approach is distributed ledger technology (DLT).

Systems built using that approach allow data to be stored in a decentralized manner across multiple computers (nodes) participating in a network.

A key feature of DLT is that it eliminates the necessity for a single center of trust.

Instead, trust is based on cryptographic protocols and a consensus mechanism that ensures that all network participants agree on a single, immutable state of the ledger [3].

Thus, distributed ledger technology offers a fundamental shift from a centralized trust model to a distributed one, responding to societal demands for more secure, transparent, and efficient public services.

Consensus mechanisms such as Proof-of-Work (PoW), Proof-of-Stake (PoS), or more modern approaches such as Directed Acyclic Graph (DAG), allow network participants to reach an agreement on the order and validity of transactions without a central authority [4].

This makes the ledger resistant to censorship, since data manipulation requires control of a majority of the network nodes.

Data immutability is another fundamental property, since each transaction is cryptographically linked to the previous ones, forming a DAG (in the simplest case, a chain) that cannot be changed retrospectively without recalculating all subsequent blocks [5].

Using DLT to manage public data previously stored in centralized cloud storage offers significant benefits.

- Increased reliability and resilience: the distributed nature of DLT makes it impossible to have a single point of failure. Even if some nodes fail, the network will continue to function, since many independent participants store copies of the ledger.
- Transparency and auditability: every participant in the network can verify transactions and verify their authenticity. This creates a completely transparent environment where any attempt at manipulation or unauthorized data change becomes obvious to all [6].
- Reduced costs: In the long run, eliminating the need for a centralized intermediary can reduce operational costs associated with maintaining and auditing the ledger.

Thus, in the context of moving personal and public data to cloud services, DLT appears not simply as a technological innovation, but as a fundamental paradigm shift.

This is a transition from a centralized model based on trust in a specific organization to a decentralized one based on cryptographic algorithms and collective verification [7].

This model provides the necessary level of security, transparency, and resilience, which is critical for registries that are supposed to serve the public good [8].

In decentralized systems, such as distributed ledgers (DL), the lack of a single, synchronized physical clock creates a fundamental problem, namely, how to determine the order of events that occur asynchronously on different nodes?

Instead of relying on unreliable physical time [9], a mechanism that provides logical time — the causal ordering of system events — is critically needed to ensure data consistency and coherence.

Logical clocks proposed by L. Lamport in [10] correctly order events and identify causal relationships between transactions, ensuring the valid operation of a DLT system.

The use of physical time (e.g., UTC system time) in distributed networks is unreliable for several reasons.

Inconsistency: Physical clocks on different nodes in the network can have clock skew due to technical failures, network delays, or malicious actions.

Small deviations in the indications of physical clocks can lead to incorrect transaction order, which is critical for logging.

Lack of cause: Physical time does not reflect cause-and-effect relationships between events.

For example, transaction A may be sent earlier than transaction B, but due to network delay, its timestamp may be later than B.

This creates paradoxes that undermine the integrity of the ledger [2].

Logical clocks are an abstract mechanism that establishes the ordering of events based on causal relationships rather than absolute time values.

A simple method for event ordering is proposed by L. Lamport in his paper [10] referred above.

It proceeds as follows:

- each node maintains a counter that is incremented when the node logs a local event;
- when a message is sent, the node includes the current value of its clock in this message;
- the recipient updates its counter by taking the maximum of its current value and the received one, and then increases the counter by one.

This mechanism guarantees that if event A is the cause of event B, then the logical time A will be less than the logical time B [10].

In DLT, this can be used to ensure that dependent transactions are processed in the correct order, preventing double-entry or other conflicts.

The Lamport clock is not only simple but also special from the point of view of category theory [11].

However, it has an important drawback, namely, different events can have the same timestamp if these events are concurrent (events that are not causally related).

This means that different transactions can have the same identifier in the DL system.

In the work [12], the authors addressed this drawback by integrating the Lamport clock algorithms with any wave algorithm.

However, this modification registers transactions with gaps in the numbering, which can also lead to difficulties during the operation of the DL system.

Thus, the question arises whether it is possible to build a logical clock that would satisfy the following conditions

- different events take different timestamps;
- if the number n is the timestamp of some event, then the number $n-1$ is also the timestamp of some other event.

To answer this question, it is necessary to have formal definitions of both the behavior of the distributed system and the logical clock that is related to this behavior.

2. Informal Presentation of Lamport Diagram

Lamport diagrams, also known as spacetime diagrams, are used to visualize the behavior (execution) of a distributed algorithm, where events and their logical sequence are mapped into logical spacetime.

They were proposed by L. Lamport in his seminal 1978 paper on logical clocks and event ordering [10].

These diagrams help to understand the partial ordering of events, which is crucial for the analysis and design of asynchronous systems.

A Lamport diagram is a planar graph.

Its vertices form a subset of 2-dimensional points with natural-valued coordinates, which model events.

The *abscissa* of an event represents the spatial dimension and corresponds to a local process (or node) of a distributed system, and its value identifies the corresponding process.

The *ordinate* of an event represents logical time, referring to the number of the event in the event flow of the local process (all local processes are considered sequential).

Thus, the event set lies within the union of vertical half-lines leading from the points of the form $(p,0)$ where p is the identifier of the corresponding local process that participates in the computation.

Events are divided into three classes depending on the type of actions that raised them.

- Local Events are events that occur within one process. For example, a calculation or an internal change of state.

- Message-Sending Events are events that occur when local processes send messages to one another.
- Message Receiving Events are events that occur when local processes receive messages from other local processes.

The edges (arrows) of a Lamport diagram connect

- either two events of one local process (lying on the half-line with the abscissa of which is determined by the process identifier), have consecutive numbers, and lead in the direction from the event with the smaller number to the event with the larger number,
- or the events of sending and receiving the same message in the direction from the sending event to the receiving event.

It is the arrows that map the cause-and-effect relationship between two events.

3. Formal Model of Lamport Diagram

In this paper, it is assumed that the set of local processes of a distributed computation is unchangeable throughout the computation.

Thus, we have a finite non-empty subset $P\subset\nat$, the elements of which are interpreted as identifiers of local processes of the distributed system.

A point $(p,n)\in\mathbb{N}\times\mathbb{N}$ represents an event.

The abscissa of such a point is interpreted as the identifier of the local process of the event, and the ordinate is interpreted as the sequential number of the event in the event stream of this process.

Unlike the traditional approach, we add to each graph vertex a frozen (such that it cannot be changed) attribute ``$act$'' that refers to the character of the event — whether it is either a local event (then the value of its attribute is ``DO''), or it is a message sending event (then the value of its attribute is ``$SEND$''), or it is a message receiving event (then the value of its attribute is ``$RECEIVE$'').

This modification is caused by the suggestion that at each moment in time, each local process is informed about what it is doing.

Remark 1. The event represented by the point $(p,n)\in\mathbb{N}\times\mathbb{N}$ is denoted by e_n^p below.

Not all points of the set $\mathbb{N}\times\mathbb{N}$ represent events of some distributed computation performed by local processes identified by elements of the set P.

Therefore, we introduce the set of events E of distributed computation.

Definition 1. The set of events of distributed computation performed by a set of local processes P is called the subset $E\subset P\times\mathbb{N}$ that satisfies the conditions

1. if $(p,n+1)\in E$, then $(p,n)\in E$ for any $p\in P$ and $n\in\mathbb{N}$;
2. for any $p\in P$ there exists $n\in\mathbb{N}$ such that $(p,n)\in E$.

Remark 2. The first condition of this definition ensures the sequence of event enumeration in the event flow of each local process, and the second one guarantees that among the computation local processes, each has at least one associated event.

To refer to the value of attribute act of an event e_n^p, the denotation $e_n^p.act$ is used.

The following statement demonstrates the possibility of replacing the second condition of Def.~1 with a simpler to check but less intuitive condition.

Proposition 1. If the condition of item 1 of the Def. 1 is fulfilled, then the condition

$$\text{for any } p \in P, \text{ is true } (p,0) \in E$$

is necessary for the condition of item 2 of the definition to be true.

Proof. We use the principle of induction to prove the implication

$$(p,n) \in E \text{ implies } (p,0) \in E.$$

Indeed, for $n=0$ the proof is trivial.

If the proposition is true for $n>0$, then by the validity of the proposition for $n+1$, taking into account the condition of item 1 of Def. 1, we have $(p,n) \in E$. The induction assumption now guarantees $(p,0) \in E$.

Thus, the application of the principle of induction ensures the validity of necessary implication. □

To model the edges of the diagram related to the transmission of a message, we note that each event of receiving a message corresponds to exactly one event of sending this message.

Additionally, at the moment of receiving the message, the receiving process knows the event of sending this message, since such information can be added to the message at the moment of sending.

At the same time, the sending p rocess does not have information about the event of receiving this message at the moment of sending the message.

That is, the modeling of edges can be done using the function

$$trigger : \{e \in E \mid e.act = RECEIVE\} \to \{e \in E \mid e.act = SEND\}.$$

Thus, we have

Definition 2. Structurally, the Lamport diagram is a triple $(P, E, trigger)$, where

- P is a nonempty finite set of natural numbers;
- $E \subset P \times \mathbb{N}\$$ is a set of computation events attributed by the set $\{DO, SEND¿, RECEIVE\}$;
- a function $trigger : \{e \in E \mid e.act = RECEIVE\} \to \{e \in E \mid e.act = SEND\}$ that describes a connection between a message receiving event and the corresponding message sending event.

However, for such a triple to truly represent a Lamport diagram, it is necessary to impose constraints on its components.

First of all, the diagram includes at least one process.

L1: The number of elements in P is a positive integer.

Further, $E \subset P \times \mathbb{N}$ must satisfy the following conditions.

L2: If $(p, n+1) \in E$, then $(p,n) \in E$ for any $p \in P$ and $n \in \mathbb{N}$.

L3: For any $p \in P$, $(p,0) \in E$.

Message transmission is assumed to occur between different processes.

L4: For any $e_n^p, e_m^q \in E$ such that $e_m^q.act = RECEIVE$, and $trigger(e_m^q) = e_n^p$, it is true that $p \neq q$.

In addition, it is necessary to prohibit the repeated receiving of a message, since it is considered that the lifetime of a message begins at the moment of its sending and ends at the moment of its receipt, and the message itself is uniquely identified by the events of its sending and receiving.

L5: For any $e_n^p, e_m^q \in E$ such that $e_n^p.act = e_m^q.act = RECEIVE¿$, and $trigger(e_n^p) = trigger(e_m^q)$, it is true that $p \neq q$.

Finally, it is generally assumed that the network is reliable, i.e., every message sent will be received eventually.

L6: For any $e_n^p \in E$ such that $e_n^p.act = SEND$, there exists $e_m^q \in E$ such that $e_m^q.act = RECEIVE$, and $trigger(e_m^q) = e_n^p$.

Summarizing, we obtain the following definition.

Definition 3 (*Lamport diagrams*). A triple $D(P, E, trigger)$, which is constructed according to Def. 1 and satisfies the constraints **L1**-**L6**, is called a Lamport diagram.

4. Causality and Logical Clocks

Lamport diagrams depict possible execution scenarios for distributed algorithms.

Just as a set of event flows defines the semantics of a sequential (single-process) algorithm, a set of Lamport diagrams can define the semantics of a distributed algorithm.

However, the events of a single-process algorithm are naturally ordered by their occurrence in the algorithm's event flow.

In contrast, for a distributed algorithm, the order can only be specified for the events of each local process separately.

4.1. "Happen Before" Relation

To order the events of a distributed algorithm execution scenario represented by a spacetime diagram, Lamport proposed using the ``happen before'' relation.

He considered this relation as a formal model of a causal relationship in distributed computations.

Lamport understood causality as follows [10]

1. each local process has the local causality relation that is determined by the flow of the events of this process;
2. each message sending event causes the receiving event of this message;
3. "happen before" relation is the least transitive relation that meets conditions of items 1 and 2.

More formally.

Definition 4. For a Lamport diagram $D = (P, E, trigger)$, the "happen before" relation is a binary relation $\rightarrow \subset E \times E$ that is the transitive closure of the relation $\cdot> \subset E \times E$ defined as follows

1. $e_n^p \cdot> e_{n+1}^p$ if $p \in P$, $n \in \mathbb{N}$ and $(p, n+1) \in E$;
2. $e' \cdot> e''$ if $e', e'' \in E$, $e''.act = RECEIVE$, and $e' = trigger(e'')$.

The "happen before" relation is transitive by definition, but it is not irreflexive in general.

An irreflexivity violation is directly interpreted as the presence of an anomaly in the algorithm being executed, namely, the presence of a lock.

The following example demonstrates this claim.

Example 1 (see [12]). Let us consider a Lamport diagram $Deadlock(P_2, E, trigger)$ where

$$P_2 = \{0,1\};$$

$$\begin{aligned} E \quad &= \{e_0^p | e_0^p.act = RECEIVE, p = 0,1\} \\ &\cup \{e_1^p | e_1^p.act = SEND, p = 0,1\} \\ &\cup \{e_n^p | e_n^p.act = DO, p = 0,1, n \geq 2\}; \end{aligned}$$

$trigger(e_0^0)=e_1^1$;

$trigger(e_0^1)=e_1^0$.

Evidently, *Deadlock* meets all constraints of Lamport diagrams and, also,

$$e_0^0 \cdot > e_1^0 \cdot > e_0^1 \cdot > e_1^1 \cdot > e_0^0.$$

Hence, $e_0^0 \rightarrow e_0^0$ that gives the violation of the "happen before" irreflexivity.

4.2. Logical Clocks

However, to determine whether the "happen before" relation for a diagram is irreflexive, one can use a tool proposed by L. Lamport in [10].

This tool is the concept of a logical clock.

Definition 5. Let $D=(P,E,trigger)$ be a Lamport diagram, T be an ordered set of ordinal type ω called a time scale. Then a logical clock is a totally defined function $C:E\rightarrow T$ such that

$$C(e')<C(e'') \text{ whenever } e'\rightarrow e''.$$

The following statement demonstrates the utility of the concept.

Proposition 2. Let $D=(P,E,trigger)$ be a Lamport diagram, then the existence of a logical clock for some time scale ensures that "happen before" relation is irreflexive.

Proof. Indeed, if $C:E\rightarrow T$ is such a clock, then for any $e\in E$ such that $e\rightarrow e$, one can obtain $C(e)<C(e)$. The last contradicts the irreflexivity of the order on T. □

More complicated is to prove the inverse statement.

Theorem. If $D(P,E,trigger)$ is a Lamport diagram with an irreflexive "happen before" relation, then there exists a logical clock $C:E\rightarrow \mathbb{N}$.

To prove the theorem, we use the following inductive construction of the required clock, which is denoted by *Lamport*.

Construction. Here we introduce the function $Lamport:E\rightarrow\mathbb{N}$, which in the general case is partially defined and, possibly, even totally undefined (for example, the function *Lamport* for the diagram in Example 1 is totally undefined).

```
def Lamport(e_n^p):
    if n=0:
        if e_n^p.act != RECEIVE:
            return 0
        else:
            return Lamport(trigger(e_n^p))
    # n>0
    if e_n^p.act != RECEIVE:
        return Lamport(e_{n-1}^p)+1
    # e_n^p.act == RECEIVE
    return max(Lamport(e_{n-1}^p), Lamport(trigger(e_n^p)))+1
```

Proof of Theorem. Firstly, we will prove that the function *Lamport* is totally defined due to the theorem's assumption, and secondly, we will verify that it meets the clock property.

So, let us take an event $e_n^p\in E$ and construct a 1-2-tree with the root labelled by e_n^p and other nodes labeled by events from E.

The construction rules are as follows

- if a node is labeled by e_0^p and $e_0^p.act \neq RECEIVE$, then this node is a tree leaf;
- if a node is labeled by e_0^p and $e_0^p.act = RECEIVE$, then add one child with the label $trigger(e_0^p)$;
- if a node is labeled by e_n^p with $n>0$ and $e_n^p.act \neq RECEIVE$, then add one child with the label e_{n-1}^p;
- if a node is labelled by e_n^p with $n>0$ and $e_n^p.act = RECEIVE$, then add two children with the labels e_{n-1}^p and $trigger(e_n^p)$.

Clear that the tree constructed with these rules represents the dependencies for computing the function $Lamport$.

The value $Lamport(e_n^p)$ is defined if the corresponding tree with the root labelled by e_n^p is finite.

So, our next step is to prove that the irreflexivity of the "happen before" relation ensures the finiteness of such a tree.

Assuming the contrary, one can obtain at least one infinite branch in the tree with nodes labeled by $e_n^p, e_{n_1}^{p_1}, \ldots, e_{n_k}^{p_k}, \ldots$.

The tree construction rules guarantee that $e_{n_1}^{p_1} \to e_n^p$ and $e_{n_{k+1}}^{p_{k+1}} \to e_{n_k}^{p_k}$ for any $k \geq 1$.

So, the finiteness of P ensures the existence of at least one $q \in P$ that occurs in the above sequence infinitely many times.

Hence, there is an infinite subsequence $e_{m_1}^q, \ldots, e_{m_k}^q, \ldots$ of the above sequence, and this subsequence satisfies the condition e^q_{m_{k+1}} rightarrow $e_{m_k}^q$ for any $k \geq 1$.

Note that if $m_k > m_1$ for $k>1$, then $e_{m_1}^q \to e_{m_k}^q$.

But by construction, $e_{m_k}^q \to e_{m_1}^q$ and $e_{m_1}^q \to e_{m_1}^q$ by transitivity of the "happen before" relation. So, we have obtained a contradiction to the irreflexivity of the "happen before" relation.

Thus, $m_k \leq m_1$ for all k.

The Pigeonhole principle guarantees that for some $k<l$, it is true $e_k^q = e_l^q$. By construction, $e_l^q \to e_k^q$, i.e. the "happen before" relation cannot be irreflexive.

Summing up this reasoning, one can claim that the constructed tree has no infinite branches.

Moreover, the algorithm from Construction is in fact an algorithm for calculating the height of this tree.

Further, e_n^p rightarrow e_m^q means that the tree for e_n^p is a subtree of e_m^q.

This ensures $Lamport(e_n^p) < Lamport(e_m^q)$. □

Corollary. The "happen before" relation of a Lamport diagram is irreflexive if and only if there exists a logical clock for this diagram.

Proof. The validity of this corollary follows directly from Prop. 2 and Theorem. □

In [11], it is shown that the Lamport clock of a diagram has some universal property (in the sense of Category Theory), and this property can be interpreted as the minimality.

Here, we give the corresponding direct fact.

Proposition 3. The logical clock $Lamport : E \to \mathbb{N}$ of a diagram $D(P, E, trigger)$ is the smallest logic clock among the logic clocks of D.

More precisely, if $C : E \to \mathbb{N}$ is a logical clock of the diagram, then

$$C(e) \geq Lamport(e) \text{ for any } e \in E.$$

Proof. Let us assume that $e \in E$ is an arbitrary event.

If $Lamport(e)=0$, then the tree for e from the proof of Theorem has height zero and $Lamport(e)=0\le C(e)$ for any logical clock C.

If $Lamport(e)>0$, then let us assume that the desired inequality is true for any event $e'\in E$ such that $Lamport(e')<Lamport(e)$. In this case, the tree for e from the proof of Theorem has nonzero height.

If $e.act\neq RECEIVE$, root of the tree has only one child e^p_{n-1}, if $e=e^p_n$, hence $Lamport(e^p_{n-1})=Lamport(e)-1$.

By assumption, $Lamport(e^p_{n-1})\le C(e^p_{n-1})$ or $Lamport(e)\le C(e^p_{n-1})+1$.

Due to the tree constructing, $e^p_{n-1}\rightarrow e$ that ensures $C(e^p_{n-1})<C(e)$ or $C(e^p_{n-1})+1\le C(e)$.

Thus, $Lamport(e)\le C(e)$ in this case.

If $e.act=RECEIVE$, root of the tree has two children e^p_{n-1}, if $e=e^p_n$, and $trigger(e)$.

Moreover,

$$Lamport(e)=\max(Lamport(e^p_{n-1}),Lamport(trigger(e)))+1. \quad (1)$$

Therefore, $Lamport(e^p_{n-1})\le C(e^p_{n-1})$ and $Lamport(trigger(e))\le C(trigger(e))$ by assumption.

Hence, we have

$$Lamport(e^p_{n-1})<C(e) \text{ and } Lamport(trigger(e))<C(e)$$

due to $e^p_{n-1}\rightarrow e$ and $trigger(e)\rightarrow e$.

Therefore,

$$\max(Lamport(e^p_{n-1}),Lamport(trigger(e)))<C(e)$$

and, due to (1) $Lamport(e)¿C(e)$.

Thus, referring to the induction principle completes the proof. □

4.3. Injective Logical Clocks

The main problem with the Lamport clock is that its timestamps cannot serve as event identifiers because different events can have the same timestamp. The following example demonstrates this.

Example 2. Let us consider a Lamport diagrsm $Example(P_2,E,trigger)$ where

- $P_2=\{0,1\}$;
- $E=\{e^p_{2k}|e^p_{2k}.act=SEND, p=0,1, k\in\mathbb{N}\}\cup\{e^p_{2k+1}|e^p_{2k+1}.act=RECEIVE, p=0,1, k\in\mathbb{N}\}$;
- $trigger(e^0_{2k+1})=e^1_{2k}$;
- $trigger(e^1_{2k+1})=e^0_{2k}$.

It is evident that $Lamport(e^p_n)=n$ for this diagram.

So, $Lamport(e^0_n)=Lamport(e^1_n)$ but $e^0_n\neq e^1_n$ for any $n\in\mathbb{N}$.

In [13], a modification of the Lamport clock algorithm was proposed, combining this algorithm with a wave algorithm.

This modification considers the set $T=\mathbb{N}\times P$ as a time scale with the following ordering

$$(n,p)<(m,q) \text{ if either } n<m \text{ or } n=m \text{ and } p<q.$$

It is evident that $(T,<)$ is a well-ordered set of the ordinal type ω.

The proposed in [12, 13] algorithm calculates the following function

```
def modifiedLamport(e^p_n):
    if n == 0:
        if e^p_n.act != RECEIVE:
```

return $(0, p)$

else: # $e_n^p.act$ == $RECEIVE$

return modifiedLamport($trigger(e_n^p)$)

else: # $n>0$

if $e_n^p.act$!= $RECEIVE$}:

m, _ = modifiedLamport(e_{n-1}^p)

return (m + 1, p)

else: # $e_n^p.act$ == $RECEIVE$

m_1, _ = modifiedLamport(e_{n-1}^p)

m_2, _ = modifiedLamport($trigger(e_n^p)$)

return $(\max(m_1, m_2) + 1, p)$

Proposition 4. The function $modifiedLamport : E \to T$ is an injective logical clock.

Proof . Note first that the function $modifiedLamport$ is a logical clock.

Indeed, by construction, $Lamport = proj_1 \circ modifiedLamport$ where $proj_1 : \mathbb{N} \times P \to \mathbb{N}$ such that $proj_1(n, p) = n$.

Hence, if $e' \to e''$ then $Lamport(e') < Lamport(e'')$ and, therefore,

$$modifiedLamport(e') < modifiedLamport(e'').$$

If $e' \neq e''$ and $Lamport(e') = Lamport(e'')$ then both $e' \to e''$ and $e'' \to e'$ are not valid.

This implies that if $e' = e_n^p$ and $e'' = e_m^q$, then $p \neq q$.

Hence, $modifiedLamport(e') \neq modifiedLamport(e'')$. □

5. Conclusions

This paper has addressed the fundamental challenge of event ordering and causality determination in asynchronous distributed systems, a problem of increasing relevance given the widespread use of Distributed Ledger Technology (DLT).

DLT requires a transition from centralized, trust-based models to decentralized systems based on collective verification and cryptographic algorithms, necessitating robust mechanisms for maintaining data consistency.

The paper has highlighted that the use of unreliable physical time (due to clock skew and its inability to reflect cause-and-effect relationships) is insufficient for ordering system transactions.

To formally analyze causal relationships in distributed computations, we have used Lamport diagrams, which visualize the execution scenarios and map events into logical spacetime.

We have developed a rigorous formal model, $D(P, E, trigger)$, for these diagrams, defining the necessary semantic constraints (see constraints **L1** – **L6** in Sec. 3.

Central to this analysis is the “happen before«"» relation, which formally captures causality based on local process flow and message passing.

Crucially, we have confirmed that the existence of a logical clock $C : E \to T$ is a necessary and sufficient condition for the irreflexivity of the “happen before” relation (see Corollary of the main theorem), thereby guaranteeing the absence of anomalies such as deadlocks.

Furthermore, the classic Lamport clock has proven to be the smallest logical clock for any diagram, demonstrating a property that can be interpreted as minimality.

While the Lamport clock correctly orders events and identifies causal relationships, its primary limitation is its lack of injectivity: different concurrent events can possess the same timestamp (Example 2).

This non-injectivity prevents the timestamp from serving as a unique event identifier, which is necessary for efficient operation within a DL system.

To overcome this drawback, we have examined a modification of the Lamport clock algorithm by combining it with a wave algorithm.

This *modifiedLamport* function operates on the specific time scale, which is not $\mathbb{N}$.

We have demonstrated that this function is an injective logical clock (Prop. 4).

This successful modification ensures that all distinct events receive different timestamps, allowing the logical time value to serve reliably as a unique identifier, thereby enhancing the integrity and functionality of DLT systems.

Declaration on Generative AI

The author(s) have not employed any Generative AI tools.

References

[1] S. Nakamoto, Bitcoin: A Peer-to-Peer Electronic Cash System, 2008. URL: https://dx.doi.org/10.2139/ssrn.3440802.

[2] S. Tern, Survey of Smart Contract Technology and Application Based on Blockchain, Open Journal of Applied Sciences, 11 (2021) 135-148. https://doi.org.10.4236/ojapps.2021.1110085.

[3] M. Crosby, P. Pattanayak, S. Verma, V. Kalyanaraman, Blockchain technology: Beyond Bitcoin, Applied Innovation, 2 (2016), 71-79.

[4] C. Yang, W. Li, Y. Zhang, A Survey of Consensus Mechanisms in Blockchain Systems, Journal of Cybersecurity and Privacy, 2 (2020), 3-21.

[5] J. Yli-Huumo, D. Ko, S. Choi, S. Park, K. Smolander, Where Is Current Research on Blockchain Technology? -- A Systematic Review, PLoS ONE, 11 (2016). https://doi.org/10.1371/journal.pone.0163477.

[6] M. Swan, Blockchain: Blueprint for a new economy. O'Reilly Media Inc, 2015.

[7] X. Xu, C. Pautasso, W. Zhang et al., A taxonomy of blockchain-based distributed ledger systems, in: Proceedings of IEEE International Conference on Software Architecture, IEEE, 2017, pp. 1-10. https://doi.org/10.1109/ICSA.2017.33

[8] D. Tapscott, A. Tapscott, Blockchain Revolution: How the Technology Behind Bitcoin is Changing Money, Business, and the World. Penguin, 2016.

[9] J. Lundelius, N. Lynch, An Upper and Lower Bounds for Clock Synchronization, Information and Control, 62 (1984), 190-204. https://doi.org/10.1016/S0019-9958(84)80033-9.

[10] L. Lamport, Time, clocks, and the ordering of events in a distributed system, CACM, vol. 21(1978), 558-565. https://doi.org/10.1145/359545.359563.

[11] G. Zholtkevych, L. Polyakova, H. El-Zein, Category Methods for Modelling Logical Time Based on the Concept of Clocks in: Information and Communication Technologies in Education, Research, and Industrial Applications, CCIS, 2019, 1007, pp. 89-101. https://doi.org/10.1007/978-3-030-13929-2_5.

[12] O. Berezovskyi, B. Melnyk, G. Zholtkevych et al, Advanced Timestamping: Synchronization and Consistency in: Public Information Systems, 15th International Conference on Advanced Computer Information Technologies (ACIT-2025), Sybenik, Croatia, IEEE, 2025, 2925, pp. 708-712. https://doi.org/10.1109/ACIT65614.2025.11185729.

[13] O. Berezovskyi, M. Terletskyi, Distributed Data Storing Based on Distributed Transaction Ledger, Bulletin of V.N. Karazin Kharkiv National University, series Mathematical modelling. Information technology. Automated control systems, 64 (2024), 6-12. https://doi.org/10.26565/2304-6201-2024-63-01.