Revista Brasileira de Informática na Educação – RBIE
Brazilian Journal of Computers in Education
(ISSN online: 2317-6121; print: 1414-5685)
https://sol.sbc.org.br/journals/index.php/rbie

Submission: 23/05/2025; 1st round notif.: 23/04/2026; New version: 25/04/2026; 2nd round notif.: 28/04/2026;
Camera ready: 13/05/2026; Edition review: 20/05/2026; Available online: 20/05/2026; Published: 20/05/2026;

Lamport Logical Clock Teaching Tool (LLCTT): A New Tool for Teaching Clock Synchronization in Distributed Systems

Diógenes Antonio Marques José
Computer Networks Laboratory (CNL)
State University of Mato Grosso (UNEMAT)
ORCID: 0000-0002-9707-6022
dioxfile@unemat.br

Bruno Hernandes Carrilho Martins
Computer Networks Laboratory (CNL)
State University of Mato Grosso (UNEMAT)
ORCID: 0009-0001-6276-3463
hernandes.bruno@unemat.br

Abstract

Distributed Systems is a core subject in Computer Science, and clock synchronization is one of its most critical topics. Among the various algorithms developed for this purpose, Lamport's logical clock stands out as a foundational concept. However, due to its abstract nature, many students struggle to fully grasp it. One of the main challenges in teaching this subject is the duality between physical time and causal ordering in real-world scenarios. To address this, we present the ***Lamport Logical Clock Teaching Tool (LLCTT)****. Developed in Python, LLCTT is designed with a two-fold approach to enhance the learning experience. First, to ensure a reliable experimental environment, the tool implements a physical synchronization module that addresses hardware failures—specifically discharged CMOS (Complementary Metal-Oxide-Semiconductor) batteries—by automatically aligning the physical date and time of laboratory nodes. Second, and primarily, LLCTT features an intuitive graphical interface that simulates Lamport's Logical Clocks, allowing users to visualize event ordering and counter increments independent of physical time. To evaluate LLCTT, we conducted multiple tests in real network environments. The results confirmed the tool's effectiveness in both correcting physical clock discrepancies and demonstrating the logical ordering of events. With LLCTT, students can clearly observe the distinction between physical time maintenance and logical causal tracking, significantly simplifying the understanding of these fundamental concepts. LLCTT has been integrated into the Distributed Systems curriculum for over a year and has proven to be an effective educational aid.*
Keywords: *Distributed Systems; Clock Synchronization; Lamport Logic Clocks; Teaching Tool; Python.*

1 Introduction

Computer networks, and consequently the *Internet*, are an excellent achievement for humanity. Thus, software running on this infrastructure is a distributed system. According to (Tanenbaum & Steen, 2023), a distributed system appears to the user as a single, coherent system. However, its components are geographically dispersed. Examples of distributed systems include *e-mail* services, name resolution services (*DNS*), databases, *peer-to-peer* networks, and messaging applications such as *WhatsApp*. Due to their relevance, distributed systems have become a core subject in Computer Science programs worldwide, with clock synchronization being one of its most critical topics.

Clock synchronization in distributed systems occurs when a process needs to know the correct time to record the occurrence of events, such as message send and receive operations. However, reaching a time agreement is challenging because there is no global clock on which all

Cite as: José & Martins. (2026). Lamport Logical Clock Teaching Tool (LLCTT): A New Tool for Teaching Clock Synchronization in Distributed Systems. Revista Brasileira de Informática na Educação, vol. 34, pp. 739–769. https://doi.org/10.5753/rbie.2026.6060.

processes can agree. As a result, many clock synchronization algorithms have been proposed over the years, such as (Gusella & Zatti, 1989), (Cristian, 1989), (Mills, 1989), and (Lamport, 1978).

Given the above, one major challenge when teaching clock synchronization is the difficulty of demonstrating a practical application of Lamport Logical Clocks (Lamport, 1978). This concept—focused on the causal ordering of events rather than actual time—is highly abstract and often difficult for students to fully grasp, unlike the concept of physical clocks that are part of their daily lives. Therefore, this article presents a didactic tool, the *Lamport Logical Clock Teaching Tool (LLCTT)*, whose primary function is to facilitate the learning of logical clocks. *LLCTT* was developed in *Python* with a graphical user interface (GUI) built in *WX-Python*. Additionally, it utilizes standard communication protocols, such as *UDP* over *IPv4* and *IPv6*.

To provide a stable environment for these theoretical simulations, *LLCTT* first addresses a common infrastructure issue in educational laboratories: the depletion of *Complementary Metal-Oxide-Semiconductor (CMOS)* batteries, which leads to incorrect clock configurations on local machines. To solve this, the tool employs a physical synchronization module based on a highest-timestamp heuristic. Consequently, when running the application, all connected computers are automatically synchronized to the highest network date and time via a single *broadcast* or *multicast* message, converging within 1 second. Its main features include:

- A user-friendly graphical interface developed in *WX-Python*;
- Native support for both *IPv4* and *IPv6* networks;
- Implementation of *Unicast*, *Broadcast*, and *Multicast* transmission methods;
- Cross-platform compatibility, installable on both *Linux* and *Windows* operating systems;
- Architecture based on core distributed systems concepts, including *Datagram (DGRAM) sockets*, *Multithreading*, and Object-Oriented Programming (OOP);
- A proprietary distribution model with trial access available for university students;
- A dedicated *GitHub* repository providing comprehensive usage and installation instructions[1].

Consequently, the main contributions of the LLCTT consist of: **1st** - Facilitating the learning of logical clocks in distributed systems, translating abstract causal concepts into an interactive and visual practical application; and **2nd** - Mitigating the real-world problem of discharged CMOS batteries in laboratory environments through a physical synchronization heuristic, thereby ensuring a reliable infrastructure for executing the logical clock simulations.

The methodology applied to develop and evaluate the LLCTT comprised three main stages:

1. **Laboratory Testing:** Performed to assess the tool's functionality, performance, and synchronization accuracy under controlled network conditions.
2. **Systematic Literature Review (SLR):** Conducted to identify existing educational software and determine whether a tool featuring LLCTT's dual-layer approach already existed.

[1] https://github.com/dioxfile/Logic_Clock_Didatic_Tool.

3. **Classroom Application:** The tool was deployed in a practical Distributed Systems class to evaluate its pedagogical effectiveness based on student experience.

The SLR results indicated that previously available tools lack the comprehensive feature set provided by *LLCTT*, making it a unique solution. Furthermore, we evaluated the proposal in a network laboratory through several trials. For instance, we synchronized different nodes with varying initial timestamps, mixing *Linux* and *Windows* operating systems. The test results demonstrated the successful alignment of the physical clocks (hardware fault mitigation) alongside the logical clocks (pedagogical execution). In this context, students can clearly observe and differentiate the physical time adjustments from each causal increment of the logical clocks through the *LLCTT* interface. Additionally, the tool was tested over two semesters in the Computer Science program at *State University of Mato Grosso (UNEMAT)*, specifically during the 2025/1 Distributed Systems classes. A *Google Forms* survey was utilized to collect feedback, and the results indicated that students highly appreciated the opportunity to explore Lamport Logical Clocks through hands-on practice, rather than relying solely on theoretical instruction.

To facilitate a clearer presentation of the content, the remainder of this article is organized as follows: Section 2 outlines the theoretical foundation of the study; Section 3 details the materials and methods, including the LLCTT testing environment, the systematic literature review, and the student feedback collection process; Section 4 discusses related work; Section 5 describes the proposed solution and its architecture; Section 6 presents the results and discussion; and finally, Section 7 concludes the article, outlining directions for future research.

2 Theoretical Foundation

This section establishes the theoretical foundations for the proposed tool for teaching clock synchronization in distributed systems. To fully understand our approach, it is essential to explore the following concepts. Furthermore, this section addresses only Lamport and Cristian's algorithms because Lamport's Algorithm is the main objective of this proposal, and Cristian's algorithm, although not the main focus, is used to implement the **RTT Ping Average** calculation that will be described in Subsection 5.2.7.

2.1 Nodes in Distributed Systems

In distributed systems, *"nodes"* represent the individual entities that make up the computing elements (Tanenbaum & Steen, 2023). These *"nodes"* can be computers *hosts*, servers, mobile devices, or any unit that participates in communication and resource sharing within the distributed system. Each *"node"* has its own processing and storage capacity, as well as the ability to communicate with other *"nodes"* through communication networks, *sockets*. Thus, *"nodes"* play a fundamental role in the coordination and execution of tasks in distributed systems.

2.2 Clock Synchronization in Distributed Systems

Clock synchronization in distributed systems is crucial in efficiently coordinating operations in a geographically dispersed environment, as approached by (Tanenbaum & Steen, 2023). In this

context, two fundamental approaches are synchronization based on physical clocks and the logical clock approach.

Physical clocks, such as those used in everyday life, seek to maintain an absolute and uniform time, but they can be affected by variations in network latency and other factors in distributed systems. Protocols such as *NTP*, proposed by (Mills, 1989), and *TSP*, proposed by (Adams et al., 2001), are used to adjust physical clocks in different locations, minimizing drift and ensuring a common notion of time. However, even with these protocols, the lag between physical clocks can still occur due to signal propagation delays and fluctuations in network latency. On the other hand, logical clocks and vector clocks proposed by (Lamport, 1978; Lamport & Melliar-Smith, 1985) prioritize event ordering over keeping an absolute count of time. They are instrumental when it is more important to establish the correct order of events, regardless of the lag between physical clocks. These logical clocks allow a more accurate representation of the temporal relationships between events in distributed systems without considering the absolute time due to the lag between the physical clocks of the *"nodes"*.

2.3 *Lamport's Logical Clocks*

The concept of *Lamport's Logical Clocks* is fundamental in distributed systems. It was developed by *Leslie Lamport* in 1978 and published in his paper "*Time, clocks, and the ordering of events in a distributed system*"[2]. This concept represents an innovative approach to record the temporal order of events in environments where the clocks of different nodes are not synchronized. Thus, instead of relying on physical clocks, logical clocks use logical timestamps to determine the precedence of events and the order in which they occur.

2.3.1 How does Lamport's Logical Clocks Work?

In logical clocks, date/time synchronization does not need to be absolute. Furthermore, if two processes do not interact, their clocks do not need to be synchronized. Thus, what happens before is taken into account, for example, two events of a process P_i, ***a*** and ***b***, where ***a*** is the sending of a message and ***b*** is the receiving of that same message. Therefore, it is equivalent to saying that $a \longrightarrow b$ (i.e., a implies b). In this way, all processes agree that event ***a*** occurs first, and then event ***b*** occurs. In this context, there are two situations:

- The first is where ***a*** and ***b*** are from the same process and ***a*** occurs before ***b***, so $a \longrightarrow b$ is true;
- And the second is where ***a*** is the event of the message to be sent by P_1, and ***b*** is the event of the same message to be received by P_2, so $a \longrightarrow b$ is true. Additionally, a message cannot be received before it is sent.

The relation between events $a \longrightarrow b$ is transitive. Therefore, $a \longrightarrow b$ and $b \longrightarrow c$, then $a \longrightarrow c$. Consequently, event ***a*** has time ***C(a)*** (which everyone agrees on). Thus, if $a \longrightarrow b$, then their clocks have times $C(a) < C(b)$. Furthermore, time ***(C)*** always runs forward. Thus, time is always positively corrected ***(+)***. Given the above, Lamport's algorithm is described as follows:

[2]Link: https://dl.acm.org/doi/10.1145/359545.359563.

1. Before some event (e.g., sending to the network and delivering to the application), the process p_i executes $C_i \longleftarrow C_i$ ***+ 1***;
2. If the process p_i sends a message ***m*** to the process p_j, it must adjust the ***timestamp*** of ***m***, ***ts(m)***, to be equal to C_i, after having executed step 1;
3. Upon receiving ***m***, P_j, sets the local counter to $C_j \longleftarrow$ ***max***$\{C_j, ts(m)\}$, after this operation step 1 is executed and the message is delivered to the application. In short, each process p_i maintains a local counter C_i.

Figure 1 presents the application of the concept of *Lamport's Logical Clocks* with three processes.

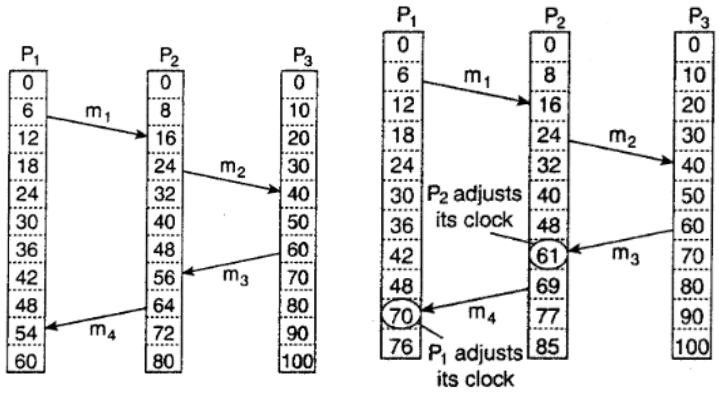


Figure 1: In (a), three processes are shown with their own clocks. The clocks run at different rates. In (b), Lamport's Algorithm Corrects the Clocks (Tanenbaum & Steen, 2023).

The Algorithm 1 presents, in a compact way, the operation of *Lamport's Logical Clocks*.

Algorithm 1 Logic Clock Lamport.

Input: *Remote Time* $ts(m_j)$ *Local Time* $ts(m_i)$

if Message Sending **then**
  $C_i = C_i + 1$;
  $ts_i = C_i$;
  Send(m_i, ts_i);
end if
if Message Receiving **then**
  (m_i, ts_i) = Receive();
  $C_j = max(ts_i, C_j) + 1$;
end if

2.4 Cristian's Algorithm

The *Cristian's Algorithm*, named in honor of *Flaviu Cristian*, is a widely used technique for clock synchronization in distributed systems. The primary function of this algorithm is to allow clocks

on different *nodes* to be adjusted to reflect a typical time, ensuring that events are ordered coherently across the environment. The *Cristian's Algorithm* offers an effective solution to mitigate the time lags and latency variations common in distributed networks.

2.4.1 How does Cristian's Algorithm Work?

Cristian's algorithm operates on a central *reference node*, known as the *"time server"*, which serves as a reliable time source. This time server periodically issues accurate timestamps to the distributed system's other *nodes*. Here is an example of how it works:

1. **Initialization:** Initially, all *nodes* in the distributed system are out of sync, and each *node* has its local clock. Thus, the time server is configured as the time authority;
2. **Time Request:** Periodically, *nodes* in the system send time requests to the time server. These requests can be made at regular intervals or as needed;
3. **Server Response:** The time server responds to time requests by providing its current timestamp, considered an accurate reference. This response includes the time the server message was sent;
4. **Adjustment Calculation:** when a *node* receives the response from the time server, it compares the received timestamp with its local clock. This allows it to calculate the difference between the server time and the *node*'s local time. This difference is known as the *"adjustment"* or *"offset"* and represents how much the *node*'s local clock needs to be adjusted to synchronize with the time server;
5. **Clock Adjustment:** based on the calculated adjustment, the *node* adjusts its local clock. However, it may involve adding or subtracting a specific amount of time to align its clock with the time server's time;
6. **Continuous Synchronization:** The request, response, offset calculation, and the clock adjustment process are repeated periodically to maintain continuous synchronization between the *nodes* and the time server.

Therefore, this algorithm is used in asynchronous systems where the round-trip time *(RTT)* between the client and server is shorter than the desired precision. Thus, the formula used to calculate the client clock is $t + \frac{RTT}{2}$, where $\boldsymbol{t}$ is the clock time returned by the server. The precision is given by $\pm(\frac{RTT}{2} - Tmin)$, where $\boldsymbol{Tmin}$ is the minimum round-trip time. If $\boldsymbol{Tmin}$ is unknown, it is assumed to be zero. The closer $\boldsymbol{RTT}$ approaches $\boldsymbol{Tmin}$, the greater the accuracy.

In addition, *Cristian's Algorithm* allows the local clocks of the *nodes* to approximate a common notion of time, reducing the skew between clocks in a distributed system. This synchronization is essential for efficiently coordinating events and transactions in distributed environments, such as database systems and network communication.

2.4.2 Cristian's Algorithm Calculation

Imagine that a client node requested the time from the server, and from the client node's point of view, the request time is 08:35:23.936. In this context, the time the message arrives at the server

is 08:40:04.025 (from the server's perspective). Therefore, the server took 60ms to process the request. After this time, the server returned the request to the client, which arrived at 08:35:24.864 (from the client's perspective). Given the above, answer the following: What time will the client set on its local clock?

To answer this question, the following calculations need to be performed:

- time sent by the client (CSH): 08:35:23.936;
- time the server received the client's request (HRC): 08:40:04.025;
- time spent by the server to process the request (TPS): 60ms;
- time returned to the client (T): HRC + TPS = 08:40:04.085;
- calculate RTT on the client: time the client received the time from the server (from client's point of view) minus CSH $\longrightarrow$ 08:35:24.864 - 08:35:23.936 $\longrightarrow$ 928ms;
- updated time on client (CUT): T + $\frac{RTT}{2}$ = 08:40:04.085 + $\frac{928ms}{2}$ $\longrightarrow$ 08:40:04.085 + 464ms = 08:40:04.549.

3 Materials and Methods

This section describes the methodology used to develop and test the proposed tool. The developed tool aims to help teach clock synchronization using the algorithms of *Lamport* and *Cristian*. In addition, a systematic review of the literature presents the primary objective of demonstrating the scarcity of tools dedicated to teaching distributed systems. Therefore, as a secondary objective, the literature review also served to compose the related work section (Section 4). Therefore, by detailing these aspects, readers will understand the process of creating and implementing the tool that we will call *Lamport Logical Clock Teaching Tool (LLCTT)*.

3.1 Hardware, Software Used, and Experiments Performed

To test the *Lamport Logical Clock Teaching Tool (LLCTT)*, the following resources were used:

- **OS:** *Linux Mint 21* with Xfce (Kernel 5.15.0-91-generic, *PC HP Elite with Intel Core i5 8100*, 4GB RAM, and *Fast Ethernet Network 10/100*) and *Windows 11 Enterprise* (VM in *Proxmox Cluster*, CPU 8 Core, *X86-64-v2-AES QEMU*, and 8GB of RAM);
- **Programming Language and Libraries:** *Python3* version 3.10.12-0ubuntu2 and libraries *Python3-wxgtk4.0* (GUI), *Ipaddr 2.2.0* (IPv4/IPv6), *Python3-netifaces* (network Interface), *Python3-pubsub* (events and subscription), and *Python3-dateutil* (date/time);
- **Experiments Performed:** *LLCTT* was installed in three *"nodes"*, as shown in Figures (5)(6)(7), being one *"node"* with *Linux Mint 21* and two *"nodes"* Virtualized in a Proxmox Cluster with *Windows 11*. Tests were performed with two types of transmissions *IPv4 Broadcast (172.168.10.0/24)*, *IPv4 Multicast (224.0.2.4)*, and *IPv6 Multicast (FF04::/8)*. In addition,

the timestamp was changed on all *"nodes"* so that they had different dates and times, ranging from the first day of **1970** to the last day of **2017**. In addition, it was specified that all *"nodes"* were synchronized to the most recent date/time.

3.2 Class Diagram

The class diagram plays a crucial role in modeling object-oriented systems, visually representing classes and their relationships. The class diagram of LLCTT is shown in Figure 2:

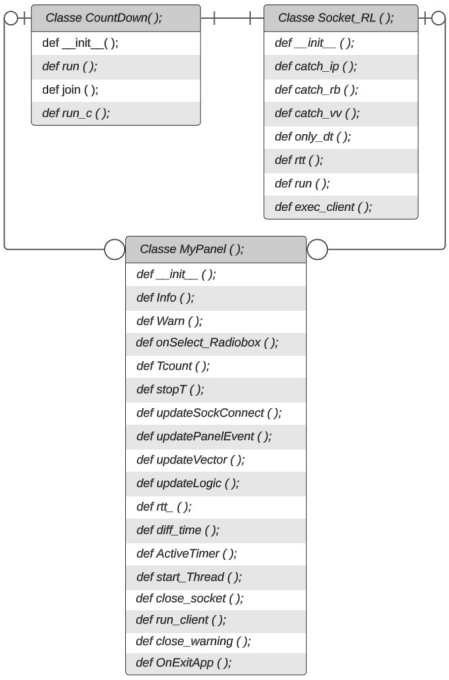


Figure 2: Class Diagram.

3.3 Systematic Literature Review

This section presents a Systematic Literature Review to identify and analyze teaching tools for clock synchronization in distributed systems, focusing on those that use *Lamport's* logical clocks. In this context, the Systematic Literature Review was used because it is a rigorous and detailed methodology that seeks to answer one or more research questions by identifying, selecting, and critically analyzing relevant studies described in the literature. Therefore, this section describes in detail the methodology, including the search procedures, study inclusion criteria, data extraction and analysis, and the results found.

3.3.1 Research Question

The research question was carefully designed to ensure the studies were comprehensive and relevant. In this case, the research question was: (RQ1)***'What are the available tools (e.g., software) that use Lamport's Logical Clocks for clock synchronization in distributed systems for educational purposes?'***.

3.3.2 Study Inclusion Criteria, Research Strategies, and Procedures

The following inclusion criteria were taken into consideration: **(1)** studies that present *Lamport's* logical clock tools/simulators for clock synchronization in distributed systems; **(2)** studies that use such tools, as per item (1), as an aid in teaching clock synchronization in distributed systems disciplines; **(3)** studies published in conference papers, journals, or symposia, or studies published in university courses (e.g., *Open Educational Resources (OER)*) through sites such as http://lasdpc.icmc.usp.br/∼ssc640/grad/ec2015/sleeping_barber/about.html; and **(4)** studies conducted between January 2004 and February 2024.

The research for this Systematic Literature Review was conducted by applying inclusion criteria to works found in the databases available at **Capes Periodicals Portal**[3], **IEEE Xplore**, and **Google Scholar**, in *Portuguese and English*. The Portuguese search string was structured as follows:**"sistemas distribuídos" AND "sincronização de relógios" AND "relógios lógicos"**. The English search string was formulated as follows: **"distributed systems" AND "clocks synchronization" AND "logical clocks"**. Therefore, these terms were combined using the Boolean operator **"AND"** to improve the precision of the search string. In addition, quotation marks were added to the strings to improve the search results. Quotation marks are used to indicate a search for an exact phrase. For example, if you search for articles containing the words "distributed systems," you would enclose the phrase in quotation marks. Without quotation marks, the search engine may return results that contain the words "systems" and "distributed" but not necessarily in the same order. With the quotation marks, the search engine will search for the exact phrase "distributed systems". Figure 3 shows the studies conducted over the last twenty years. It is notable that, over time, the number of studies in this area has decreased.

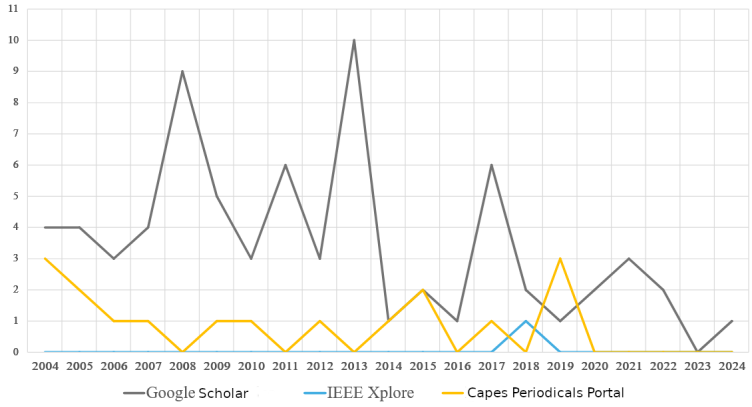


Figure 3: Studies Published from 2004 to 2024.

3.3.3 Results

The results from the selected databases yielded 89 publications in the first analysis stage (Table 1). Also, four articles were removed in this stage because they were duplicates or citations. The second stage of analysis was based on article titles, so 24 were selected. The articles' abstracts, introductions, and conclusions were analyzed, leaving only 14. Finally, in the fourth and last

[3]Institution that Fosters Scientific Research in Brazil, CAPES Portal.

stage, the analysis considered the inclusion criteria, leaving no articles as shown in Figure 4 and Table 1.

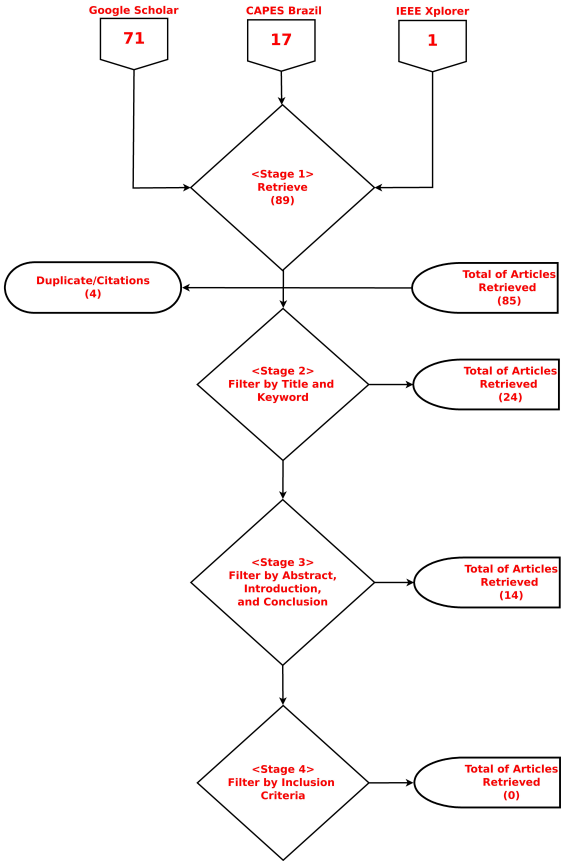


Figure 4: Flowchart of Article Selection Stages.

Table 1: Search Period and Results Found.

Language	***Google Academic***	***IEEE Xplore***	***Periodicals Capes***	***Period of Research***
Portuguese	13	0	0	2004 to Mar. 2024
English	58	1	17	2004 to Mar. 2024

Given this perception, it was necessary to conduct additional research, which was carried out freely in *forums*, *blogs*, and code repositories (e.g., *GitHub*) to identify works aligned with the inclusion criteria. Thus, this new search resulted in the articles that comprise the related work section, Section 4.

3.4 LLCTT Classroom Assessment Methodology

The LLCTT was implemented in a classroom setting during the 2025/1 term as part of the Distributed Systems course in the Computer Science program at *State University of Mato Grosso*

(UNEMAT). The practical session involving LLCTT took place one week after students were introduced to the concepts of clock synchronization and Lamport's logical clocks. Seventeen students participated in the three-hour session, during which they actively used the LLCTT. Following the activity, students were invited to evaluate the tool by completing a questionnaire administered via Google Forms.

The questionnaire consisted of five targeted questions to assess the educational effectiveness and usability of the tool. Sample questions included: *What did you think of the graphical interface?*, *Does LLCTT facilitate the learning of distributed systems in general?*, *Does the tool aid in understanding logical clocks?*, *Does the process and logical clock panel help clarify Lamport's algorithm?*, and *Which Distributed Systems concepts does LLCTT help you understand more clearly?*.

Consequently, the results of this evaluation are discussed in Section 6.3.

4 Related Work

This section presents a compilation of tools that simulate Lamport's logic clocks.

In (Rodrigues et al., 2017), an application called *Algorithm for Synchronization of Physical Clocks in Distributed Systems* is presented. In this work, the authors present an application based on Lamport's local time concept to synchronize physical clocks in a computer network. The application addresses clock synchronization and is an educational tool for learning about distributed systems and clock synchronization. This work's contributions include the development of a clock synchronization algorithm for distributed systems and the implementation of a Python application with an emphasis on multithreading and object-oriented programming. In addition, the following work demonstrates the practical application of synchronization concepts in distributed systems to solve real-world problems. Despite all these advantages, the proposal in question is limited by not providing multicast, available in *IPv4* and *IPv6*. Furthermore, it does not work on *Windows* systems.

(Taylor & Stewart, 2016), present a simulator of Lamport logic and vector clocks called *Lamport and Vector Logical Clocks*. The proposal in question was developed in **C++** and uses the ***OpenMPI*** library to facilitate simulation. The simulation allows users to understand the operation and practical application of Lamport's logical and vector clocks in distributed scenarios. However, it is important to note that this work uses the command line as its main interface. Nevertheless, this may require some prior technical knowledge of the command language. In addition, the simulator can only be used on local nodes because it does not support network communication, for example, *sockets* or *RPC*.

The application proposed in (Barcellos, 2023) is a *Python* simulator that synchronizes logical clocks using Lamport's algorithm. In this context, the simulator allows the user to specify the number of processes and generate logical times for each process with random increments. The simulator also allows the user to simulate communication events between processes, with logical times updated according to Lamport's algorithm. However, the proposal in question does not use a real network for communication and is limited to a local simulation.

In this study, developed by (Antunes, 2014), a simulator is proposed to organize events (messages) between processes of a distributed system via logical clocks. The simulator was created with the *Java* programming language, using the *Eclipse IDE* and the *WindowBuilder* extension. The application has a graphical interface that performs a simulation representative of Lamport's algorithm. It is fascinating for conceptual classes of distributed systems. However, it does not implement real network communication mechanisms, for example, exchanging messages through *sockets*, and functions only as a local simulator.

(Silva, 2020), presents a didactic material to facilitate the understanding of *Lamport's Logic Clocks*. This study offers clear instructions on how to understand Lamport's algorithm, in addition to providing examples and a source code in **C** that implements the concepts of *Lamport's Logic Clocks*. However, the proposal, despite helping beginners to understand the concept of *Lamport's Logic Clocks*, is minimal, having no graphical interface and no communication via *sockets* or *RPC*.

In (Santana, 2017), an application called *Simulation of the Lamport's Algorithm in JAVA* is presented. This application is based on Lamport's concepts and provides a graphical interface that simulates message exchange between processes. In this way, the user can select a specific number of processes to exchange messages. Therefore, the simulation adjusts the time of any incorrect process. In addition, it is possible to define when each process will receive and send messages. However, the application lacks more details, for example, a more detailed manual and the use of communication via *sockets*, because, without this, there is no way for the user to use the application without studying the code. It also limits execution to the local environment, without network interaction.

In (Luzia, 2004), a study on distributed algorithms for the consensus problem is proposed, focusing mainly on undergraduate students. It contextualizes the consensus problem in distributed systems, describes similar algorithms, and proposes a detailed study of the *Fundamental Algorithm* defined by *Pease, Shostak, and Lamport*. In addition, it presents a Java architecture for implementing these algorithms, including an example implementation of the Fundamental Algorithm. However, despite using an *API* for *socket* communication, the work does not present network tests; it only comments. Consequently, no graphical interface is presented, *GUI* that would facilitate the use of the proposal.

In (Whittaker, 2015), a simulator of *Lamport's Logical Clocks* in *Python* is proposed. The simulator provides a function that receives a list of functions, each representing a process. The functions use an instance that supports three methods (e.g., `clock()`, `recv()`, and `send()`) and return a function that can be invoked to plot the space-time graphs used to visualize the logical clocks. This proposal shows how *Lamport's Logical Clocks* work exactly as in the article published at: https://dl.acm.org/doi/pdf/10.1145/359545.359563. However, despite being developed in Python, it does not have an interactive graphical interface and runs only locally, without network interaction.

The work developed by (Tran, 2018) presents a simulator of *Lamport's Logical Clocks* together with a resource allocation algorithm in a simulated distributed environment. In this way, to exchange messages, each process's clock is identified by a unique ID and controlled by an order number. The system uses *MulticastSocket* and *DatagramPacket* from *Java* to simulate interprocess communication. In addition, an event class was defined to handle different events. There-

fore, a master *Thread* tracks the messages sent between participants. The work also extends case handling to include events related to requests, responses, and request acknowledgment, using distributed mutual exclusion. However, even though the simulator uses *MulticastSocket* and *DatagramPacket* from *Java*, it only exchanges messages between local processes on the same node. Regarding teaching, the simulator lacks a more user-friendly interface to facilitate the learning of distributed systems and could also expand communication to a real network environment.

In (Tedesco, 2019) an implementation of the *Berkeley* algorithm called *Clock Synchronization with the Berkeley Algorithm and Java RMI* is presented. The *Berkeley*[4] algorithm operates by querying each computer, checking their clock values, and consequently calculating the average of the collected data, instructing each machine to adjust its clock, either forward or backward, as necessary. The proposed implementation involves a client and at least two servers on different machines. Therefore, to run the system, it is necessary to correctly configure the environment variable (e.g., `.;%JAVA_HOME%\lib`), inform the IP of the two servers in the corresponding class, start the system, and run the class from the generated JAR file. Everything is done via the command line in this proposal because there is no *GUI* to interact with the application.

Beyond clock synchronization tools, solutions such as Panda (Cândido et al., 2021) have addressed other educational domains while sharing LLCTT's commitment to user-friendly interfaces and real-user validation. However, no existing tool offers LLCTT's dual-layer approach that integrates physical clock synchronization with the pedagogical demonstration of Lamport's logical clocks over real IPv4/IPv6 networks.

Table 2: Comparative Evaluation Between LLCTT and Other Proposals in the Literature.

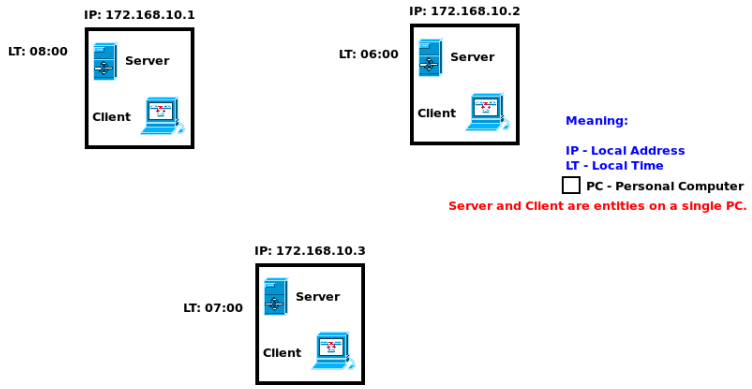
Characteristics	***LLCTT***	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***
Documentation	Yes	No	Yes	Yes	Yes	No	No	No	Yes	Yes	Yes
Programming Language	Python	Python	C++	Python	Java	C	Java	Java	Python	Java	Java
Graphical User Interface	Yes	Yes	No	No	Yes	No	Yes	No	No	No	No
Windows Operating System	Yes	No	No	No	Yes	Yes	Yes	Yes	?	Yes	Yes
Linux Operating System	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	?	Yes	Yes
Socket	Yes	Yes	No	No	No	No	No	Yes (API)	No	No	Java RMI
Real Network	Yes	Yes	No	No	No	No	No	?	No	No	No
Unicast Transmission	Yes	Yes	No	No	No	No	No	?	No	No	No
Broadcast Transmission	Yes	Yes	No	No	No	No	No	?	No	No	No
Multicast Transmission	Yes	No	No	No	No	No	No	?	No	No	No
IPv4	Yes	Yes	No	No	No	No	No	?	No	No	No
IPv6	Yes	No	No	No	No	No	No	?	No	No	No
Threads	Yes	Yes	No	No	No	No	No	Yes	No	No	No

To summarize this analysis, Table 2 presents a feature comparison between *LLCTT* and other existing proposals. While traditional simulators are excellent for demonstrating the abstract causal ordering of events locally (in-memory), they often isolate the student from the complexities of real network communication. The primary distinction of *LLCTT* is its architectural choice to bridge this gap: by implementing actual UDP/IPv4/IPv6 sockets and multicast transmissions, *LLCTT* allows students to experience logical synchronization not just as a mathematical abstraction, but as a mechanism operating over a tangible, distributed infrastructure.

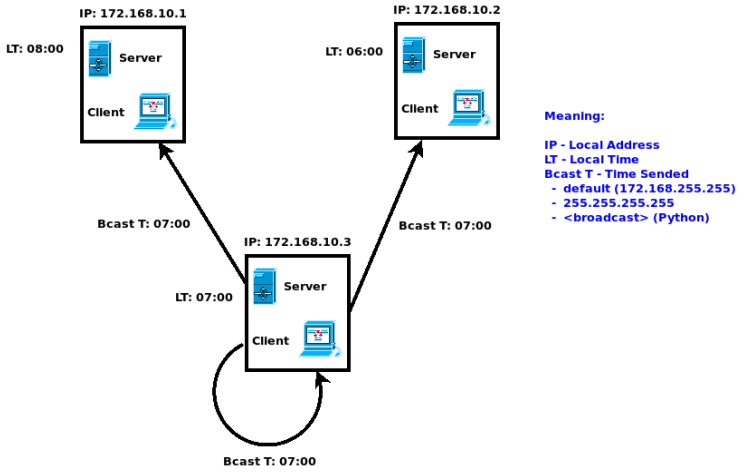
[4] https://en.wikipedia.org/wiki/Berkeley_algorithm.

5 Proposal Description

This section presents a detailed description of LLCTT. In this proposal, the *Lamport's Logic Clocks* Algorithm was employed to solve the problem of a depleted *Complementary Metal-Oxide-Semiconductor (CMOS)* battery, resulting in the incorrect configuration of the physical clocks on the computers. Consequently, when running the application and broadcasting a message over the network, all connected computers will automatically synchronize to the highest date and time.



(a) Scenario with Three Computers.



(b) Sending Messages via Broadcast in the Proposed Scenario.

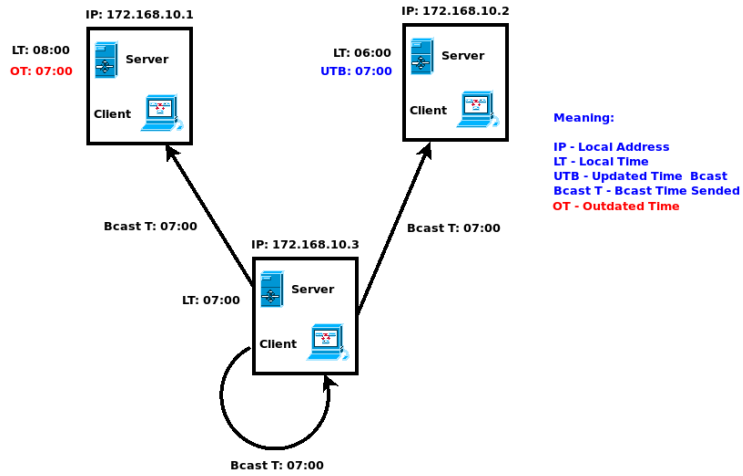
Figure 5: In (a), a Scenario with Three Nodes with Physical Clocks Running at Different Rates is Presented. In (b), the first round of message sending is presented with Node 172.168.10.3, which sends its date/time to the network.

In this context, as shown in Figure 5, to achieve synchronization across the network, only one message is sent via *broadcast* or *multicast*. It should be noted that the heuristic of adopting the highest timestamp for physical synchronization was specifically designed for controlled educational laboratory environments, where the absence of malicious nodes (*Byzantine faults*) or hardware anomalies projecting dates into an unrealistic future is assumed. In production networks, robust time intersection protocols, such as *NTP*, would be required. However, for the didactic

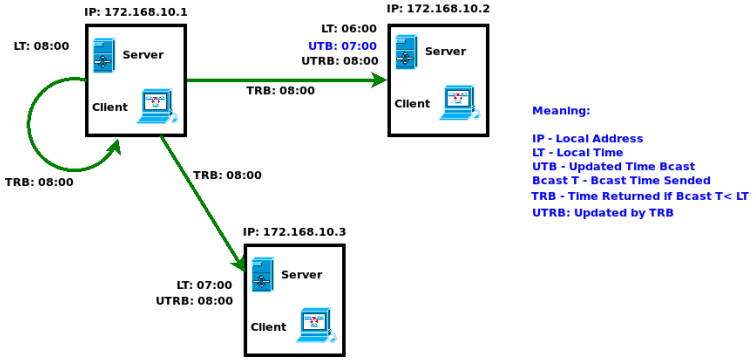
scope of the *LLCTT*, this simplification ensures the rapid convergence (1 second) necessary to start the class.

5.1 Application Operation

This tool works as follows. Imagine a scenario with three nodes, as illustrated in Figure 5. In this scenario, the node with the *IP address 172.168.10.3* sends a message with Date/Time via *broadcast* to the network as shown in Figure 5(b). Thus, when nodes **172.168.10.1** and **172.168.10.2** receive this message, they will behave as illustrated in Figure 6(a). As shown in Figure 6(a), the node with the IP address *172.168.10.2* updates its local time. However, node *172.168.10.1* does not. It happens because the time on node *172.168.10.1* is more up to date. Consequently, node *172.168.10.1* will broadcast a message to the network containing the most current time. Therefore, all other computers will update their clocks with this message. Thus, full synchronization occurs, Figure 6(b).



(a) Receipt of the Synchronization Message by Nodes 172.168.10.1 and 172.168.10.2.



(b) Return of the Most Updated Date/Time, Performed by Node 172.168.10.1.

Figure 6: In (a), the Date/Time Being Received by the Nodes is Presented, 172.168.10.1 and 172.168.10.2. In (b), the Return Via Broadcast to the Network by Node 172.168.10.1 is Presented, with Its Updated Date/Time.

Why was it decided to return a message with the most current time via *broadcast/multicast*? The answer is that if the message were used via *unicast*, for example, returning only to the sender of the message, the other machines that updated their Dates/Times with the first message would not receive the most current Date/Time, as illustrated in Figure 7.

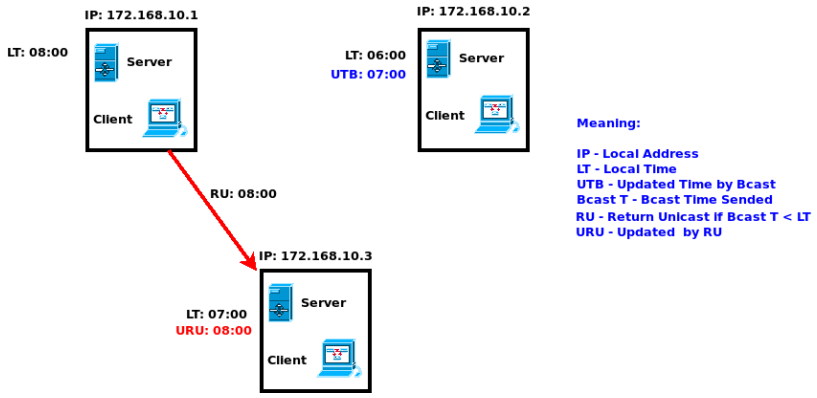


Figure 7: Illustration of Why Sending/Returning Messages via Unicast Does Not Work in *LLCTT*. This Way, Only Host 172.168.10.3 Would Update Its Local Clock.

5.2 Graphical User Interface – GUI

This section presents the *LLCTT graphical user interface (GUI)* developed with *WXPython-4.0*. In addition, each component of the *LLCTT GUI* is explained in detail.

5.2.1 Transmission and Connection Method

Figure 8 shows that LLCTT can create *DGRAM (UDP)* sockets for IPv4 (Unicast, Multicast, and Broadcast) and IPv6 (Unicast and Multicast). For example, a valid combination would be Transmission Method = *Broadcast*, Local Server IP = 0.0.0.0, Remote IP = $<$ *broadcast* $>$, and Local/Remote Port = 10001 (e.g., if you have a *firewall*, this port must be released). Furthermore, any communication port can be used as long as no other application is using it. Thus, after this configuration, when clicking on the *Bind IP/Port* button, for example, in the *Local Socket Assigned (Local)* panel (Figure 9) it is possible to see the connected *socket* (tuple), for example: `Connection in ('0.0.0.0', 10001)`.

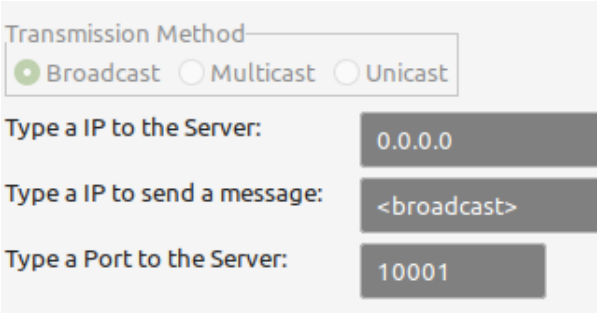


Figure 8: Transmission Method Panel, Local Server IP, Remote Server IP (Message to Send) and Local/Remote Server Port.

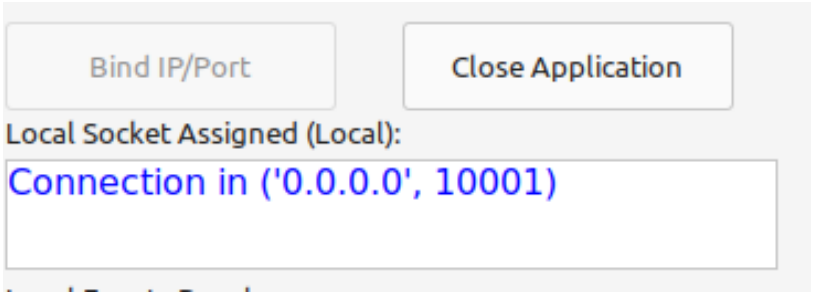


Figure 9: Bind IP/Port and Close Application buttons, and Text Box with Local Socket Assigned (Local).

5.2.2 Local Events Panel

The local events panel can display up to six different types of events, Figure 10. For example:

1 One that occurs in the application, sending messages from it to itself and other computers on the network. Example: **The process at ('172.168.20.21', 52952) says, date/time: 11/08/2023 14:31:00.701633**. It shows that the local process on *socket* ('172.168.20.21', 52952) sent the timestamp to itself and the network;

2 Return via *Multicast/Broadcast/Unicast*. This occurs when the local process returns the most recent date/time to the remote process(es) (other applications on the network). Example: **Send R:M/B/U...**;

3 Update via *Multicast/Broadcast/Unicast*. The event occurs when the local application synchronizes its date and time through some remote process via a return message. Example: **Update date/time from ('172.168.20.96', 52643) by R:M/B/U**;

4 Update via *Unicast*. It occurs when the local application synchronizes its date and time through some remote process via the Unicast Transmission Method. Example: **Update date/time from ('172.168.20.96', 52683) by unicast**;

5 Update via *Broadcast*. This event occurs when the local application synchronizes its Date/-Time through some remote process via the Broadcast Transmission Method. Example: **Update date/time from ('172.168.20.96',33643) by broadcast**;

6 Update via *Multicast*. This event occurs when the local application synchronizes its Date/-Time through some remote process via the Multicast Transmission Method. Example: **Update date/time from ('172.168.20.96', 62643) by multicast**.

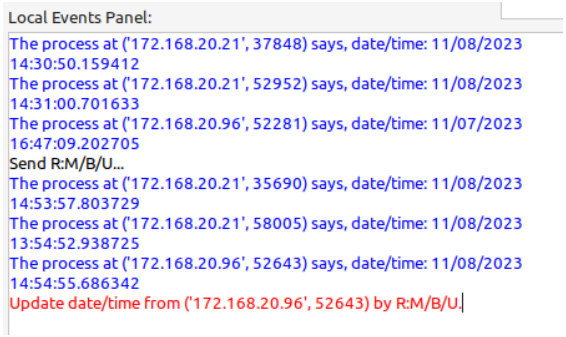


Figure 10: Panel: Local Events.

5.2.3 Hosts and Processes

In the *Hosts* and Processes Panel, Figure 11 displays the remote and local processes connected to LLCTT. For example, the local process *default server IP 0.0.0.0*, local process *IP 172.168.20.21*, and remote process *IP 172.168.20.96*.

Figure 11: Panel of Processes Connected to the LLCTT Application.

5.2.4 Lamport Logical Clock Panel

The logical clock panel, Figure 12, displays the number of events that occurred in the local application (e.g., local IP), in the case of sending messages to the network (e.g., a single application on the network) and also all the events coming from remote applications that contacted the current process, in the case of more than one application on the network.

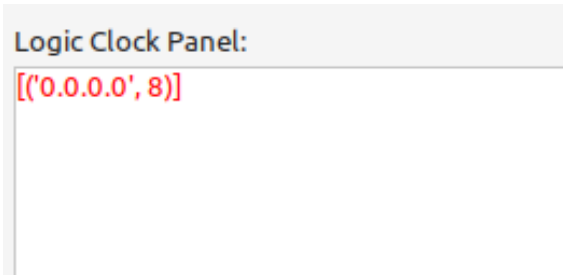


Figure 12: Lamport Logical Clock Panel Is Displaying a Tuple with the Default/Local Process and the Number of Events Occurred/Recorded in This Instance of the LLCTT Application. For example, Sending to Network and, Receiving, Delivery to Local Application.

For example, if two processes p_i and p_j, on different nodes, exchange messages (e.g., events a and b) and if the logical clock of the sending process is 8 (e.g., $C_i == 8$) this process will increment its clock by 1 (e.g., $C_i = C_i + 1$), $C_i == 9$, and after that, it will match the timestamp (e.g., date/time) of the message to be sent to the value of the logical clock C_i (e.g., $ts_i(m_i) = C_i$) and, after that, it will send the message to process p_j. Thus, process p_j will perform the following calculation upon receiving the message: $C_j = MAX\{C_j, ts_i(m_i)\} + 1$. Thus, if the logical clock of C_j is equal to 5, then the value of C_j will be $= MAX\{5,9\} + 1 \Rightarrow C_j = 10$.

5.2.5 Date/Time Display (Local Physical Clock)

The Date/Time display, Figure 13, displays the local time configured on the application's node. Thus, the application will instantly update the display if the user changes the local Date/Time on the *OS*.

Figure 13: LLCTT Date/Time Display. Local Time Is Stored In The CMOS Battery And Can Be Changed In The BIOS, In The Operating System, And Through The LLCTT.

5.2.6 Time Difference Panel

The Time Difference Panel displays the difference between the Date/Time that was updated/synchronized in the local application and the Date/Time received from a remote application. For example, if the current application updates its Date/Time from another application, it will display the time difference between the local Date/Time and the remote Date/Time in this Panel, as shown in Figure 14.

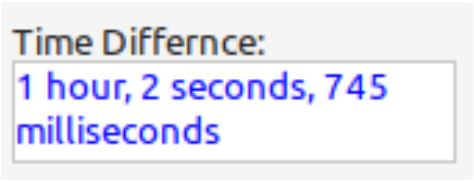


Figure 14: Time Difference Panel.

5.2.7 Average Ping RTT Delay Panel

The *Ping RTT* Average Delay Panel displays the delay used to adjust the Date/Time received from remote processes, Figure 15. This method is based on the method of (Cristian, 1989). For example:

1. The Local Process sends its timestamp to the network at time t_0;
2. All Remote Processes receive this timestamp. Thus, if a remote process has a more current time, it will return a message to the network with its timestamp $\boldsymbol{T}$;

3. The Local Process, with an outdated timestamp, receives the response at time t_1 and then adjusts its time to $T + \frac{RTT}{2}$, where $RTT = [(t_1 - t_0) + RTT\ Ping\ Average]$.

Therefore, to find the *RTT Ping Average*, the **ping** program was used. Thus, upon receiving the updated Date/Time from a remote application, the current application, "Local Process", executes the **ping** program in the direction of the remote application (e.g., $ `ping Remote IP`). Thus, **ping** returns the average *RTT* time (e.g., *rtt min/**avg**/max/mdev = 0.994/**1.021**/1.048/0.027 ms*, for example, **RTT Ping Average = 1.021 ms**. After that, *Cristian*'s method is applied as previously described, $T + \frac{RTT}{2}$. That's why this method is called *RTT Ping Average*, Figure 15.

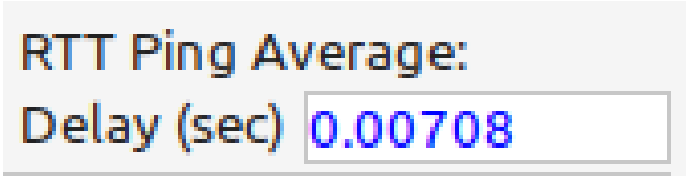


Figure 15: RTT Ping Average Panel.

5.2.8 *Message Sending System*

The Message Sending System consists of three buttons: **Automatic Send Message** Button, **Stop: Automatic Message** Button, and **Manual Send Message** Button. Thus, messages can be sent automatically or manually (Figure 16). In automatic sending mode, a message is sent to the network every 3 seconds.

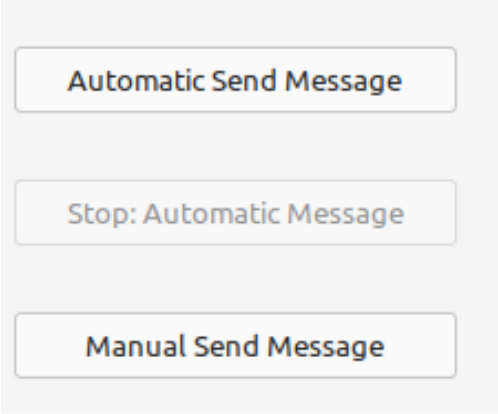


Figure 16: LLCTT Messaging System.

5.2.9 *LLCTT Graphical Interface Overview*

Figures 17 and 18 show the complete GUI running on two different operating systems, Linux and Windows. Consequently, Figure 17 shows the LLCTT graphical interface, all components, running in a Linux environment, and Figure 18 shows the graphical interface of LLCTT running on the *Microsoft Windows 11* Operating System using *VMware WorkStation 17 Player*.

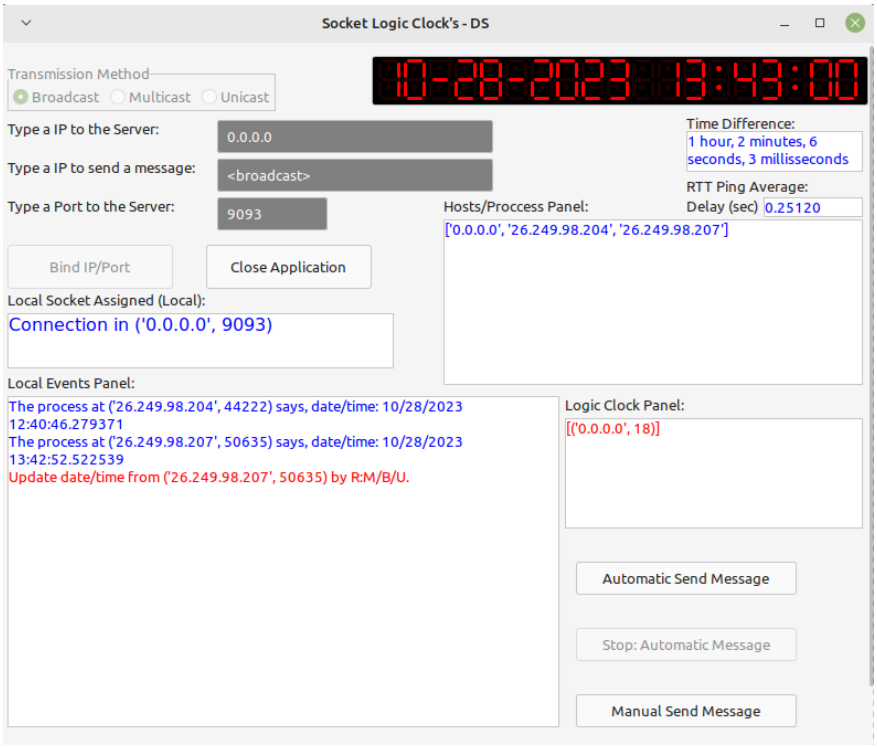


Figure 17: Complete Graphical Interface of LLCTT in Linux Environment.

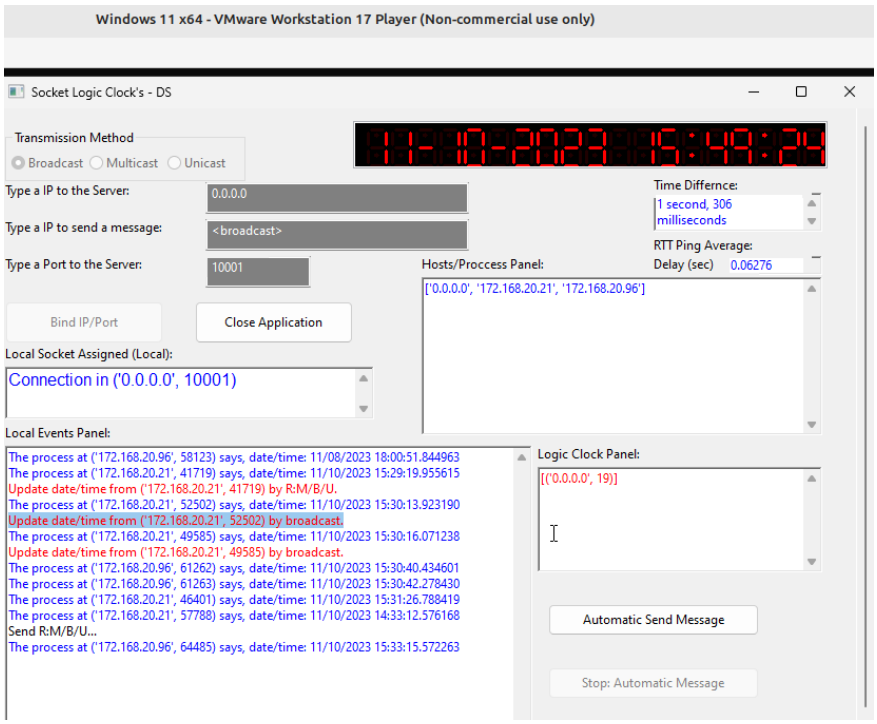


Figure 18: Complete LLCTT Graphical Interface in Windows Environment Via VMware Workstation 17 Player.

6 Results and Discussion

This section presents the results of the tests applied to *LLCTT*. The test environment configuration consisted of three machines: two virtual machines running *Windows 11* in English (EN-US) on a *Proxmox cluster*, and one physical machine running *Linux Mint 21.1* in Portuguese (PT-BR). Access to the *Windows 11* VMs was performed using the *AnyDesk* tool. The tests were divided into two parts as follows (as described in Section 3):

1. **Broadcast:** Sending messages from one machine to all, multipoint, using IPv4;
2. **Multicast:** Sending messages to a specific group, multipoint, using IPv4 (224.0.2.4) and IPv6 (FF04::/8).

It is important to highlight that, regardless of the synchronization state of the physical clocks, any message transmitted or received over the network automatically triggers the execution of Lamport's algorithm. Specifically, upon receiving a message m, the algorithm updates the local counter using the rule $C_j \leftarrow \max(C_j, ts(m))$, treating the packet as a standard communication event. Furthermore, every machine that sends a message to the network increments its logical counter ($C_i \leftarrow C_i + 1$) and includes the updated logical timestamp in the message payload, effectively performing *piggybacking*.

6.1 Broadcast

The Broadcast test performed with IPv4 used three machines, as shown in Figure 19. Besides that, within this context, the broadcast is a particular case in this experiment because it is only possible with the IPv4 protocol, since the IPv6 protocol no longer supports this type of transmission, as it has become obsolete in today's Internet.

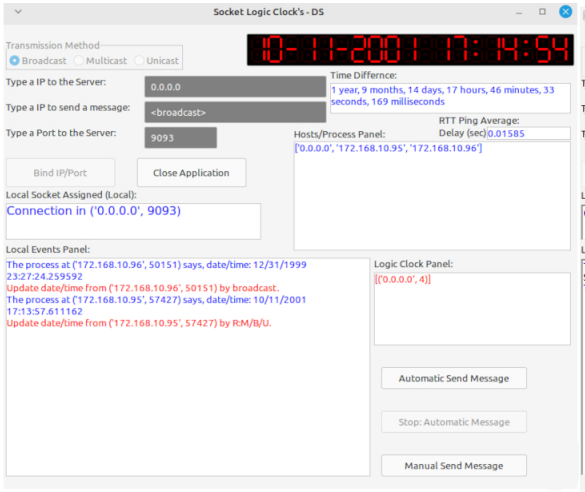
6.1.1 Broadcast IPv4

The machines were configured as follows:

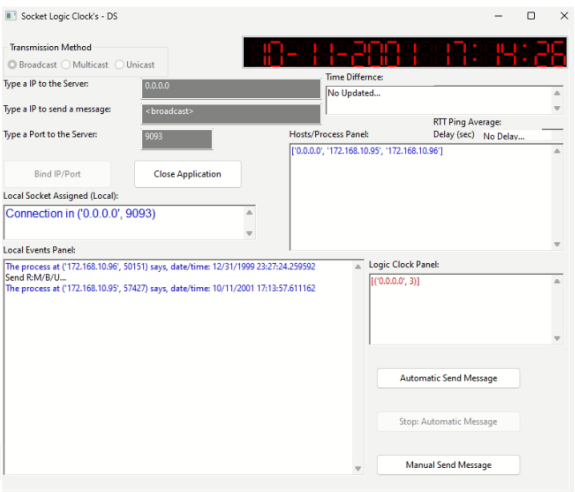
- **Machine 1:** Operating System Linux in Portuguese (PT-BR), Environment Physical, IP 172.168.10.14/24, Date and Time April 25/1979 at 01:20;
- **Machine 2:** Operating System Windows in English (EN-US), Environment Virtual, IP 172.168.10.95/24, Date and Time October 11/2001, at 5:12;
- **Machine 3:** Operating System Windows in English (EN-US), Environment Virtual, IP 172.168.10.96/24, Date and Time December 31/1999 at 23:26.

The test with *Broadcast* IPv4 has several characteristics, as shown in Figure 19(a)(b)(c). For example, in the fields "*Type an IP to the Server:* (e.g., 0.0.0.0)" and "*Type an IP to send a message:* (e.g.,)" it is not necessary to enter the IPs manually because *LLCTT* automatically uses the IPv4 of the operating system's network interface. Therefore, in this test, **Machine 3 (Windows - IP: 172.168.10.96)** sends the first message and, consequently, all other connected machines

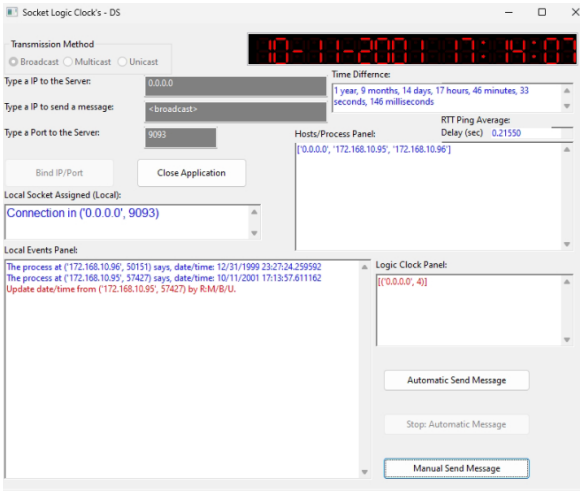
show the following message in the local events panel: "*The process at ('172.168.10.96', 50151) says, date/time 12/31/1999 23:27:24.259592*". This message contains the sender's IP, socket, and date/time. Thus, after all other machines receive the message and the network converges, their logical and physical clocks, machines 1, 2, and 3, respectively, will be updated, Figure 19.



(a) Executing LLCTT in a Linux Environment with IPv4 Broadcast, Machine 1.



(b) Execution of LLCTT in Windows Environment with IPv4 Broadcast, Machine 2.



(c) Execution of LLCTT in Windows Environment with IPv4 Broadcast, Machine 3.

Figure 19: In (a), the receipt of a message by Machine 1, Linux, sent by Machine 3, Windows, is shown. In (b), this message is received by Machine 2 (Windows); consequently, the machine automatically triggers a response to the network via **Send R:M/B/U.**... In (c), the receipt of a return message (e.g., **Update date/time from ('172.168.10.95', 57427) by R:M/B/U.**) sent by Machine 2 is shown.

In this context, the execution proceeded as follows: **Machine 3 (Windows - IP: 172.168.10.96)** sends a message containing its logical and physical timestamps to the network (e.g., Logical Clock **(1)** and Physical Clock 12/31/1999 23:26). Upon receiving this message, **Machine 1 (Linux - IP:**

172.168.10.14) updates its logical and physical clocks to **(2)** and 12/31/1999 23:26, respectively. Similarly, **Machine 2 (Windows - IP: 172.168.10.95)** increments its logical clock to **(2)**. Consequently, **Machine 2** automatically triggers a synchronization response to the network (indicated via *Send R:M/B/U...*).

Consequently, **Machine 2 (Windows - IP: 172.168.10.95)** will have its logical clock with the following timestamp **(3)**. Therefore, machines **1 (Linux - IP: 172.168.10.14)** and **3 (Windows - IP: 172.168.10.96)** upon receiving the message returned by **Machine 2 (Windows - IP: 172.168.10.95)** will update their logical and physical clocks to **(4)** and 10/11/2001 17:14. Thus, machines 1, 2, and 3 will have the following logical/physical timestamps, respectively, **(4)/10/11/2001 17:14**, (**3)/10/11/2001 17:14** and **(4)/10/11/2001 17:14**, Figure 19[5].

6.2 Multicast Synchronization Tests

The Multicast tests were performed with three machines and divided into two categories: IPv4 and IPv6.

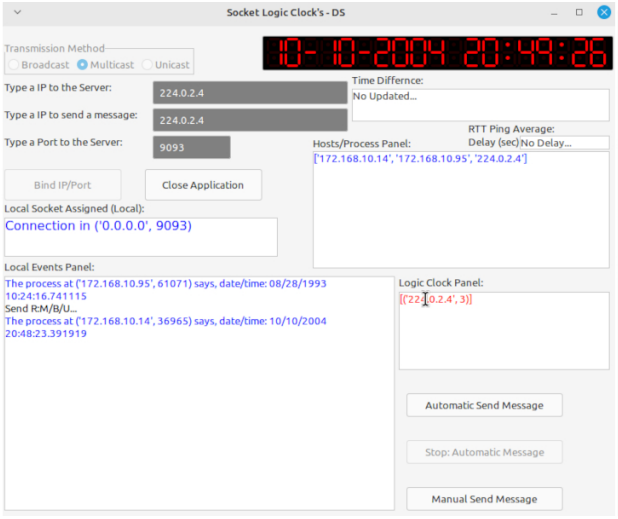
6.2.1 IPv4 Multicast

For IPv4 *Multicast*, the machines were configured as follows:

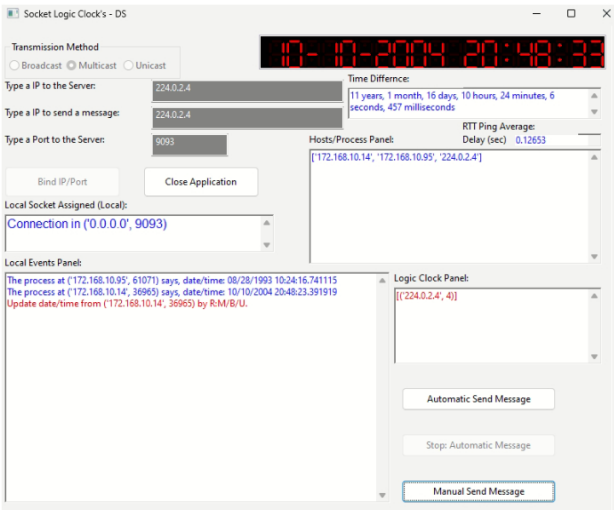
- **Machine 1:** Operating System Linux in Portuguese (PT-BR), Environment Physical, IP 172.168.10.14/24 and Group IP 224.0.2.4, Date and Time October 10/2004 at 8:45;
- **Machine 2:** Operating System Windows in English (EN-US), Environment Virtual, IP 172.168.10.95/24 and Group IP 224.0.2.4, Date and Time August 28/1993 at 10:22;
- **Machine 3:** Operating System Windows in English (EN-US), Environment Virtual, IP 172.168.10.96/24 and Group IP 224.0.2.4, Date and Time January 22/1981 at 05:47.

To make an IPv4 Multicast connection, it is necessary to enter the multicast group IP in both the Server IP field and the Recipient IP field (Figure 20). After this configuration, *LLCTT* will automatically use the IPv4 configured on the machine to establish the connection and assign it to the group (Figure 20).

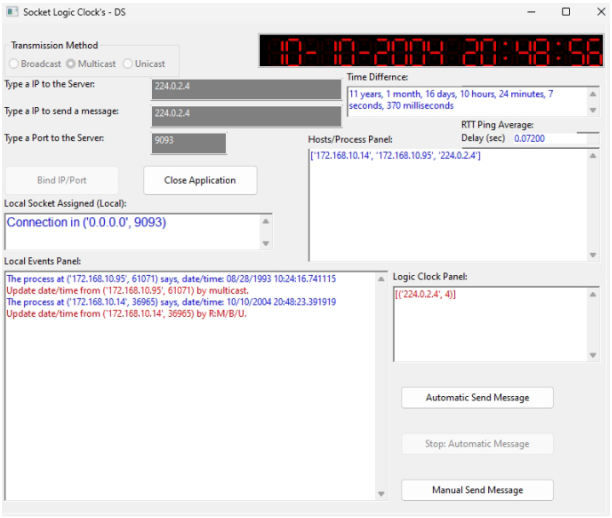
[5]The difference in seconds in the figures is due to the time taken to take the *print screen* on different machines.



(a) Executing LLCTT in a Linux Environment with IPv4 Multicast, Machine 1.



(b) Execution of LLCTT in Windows Environment with IPv4 Multicast, Machine 2.



(c) Execution of LLCTT in Windows Environment with IPv4 Multicast, Machine 3.

Figure 20: In (a), the reception of a message sent by Machine 2, Windows, is shown. Furthermore, upon receiving the message, Machine 1 automatically returns a **Send R:M/B/U...** message to the network. In (b), Machine 2 (e.g., Windows) is shown sending the first message to the network. In (c), we see the receipt of a return message (e.g., **Update date/time from ('172.168.10.14', 36965) by R:M/B/U.**), which was sent by Machine 1.

In this experiment, **Machine 2 (Windows - IP: 172.168.10.95)** sends the first message. Accordingly, in this context, the message: "*The process at ('172.168.10.95', 61071) says, date/time 08/28/1993 10:24:16.741115*" is displayed in the event pane of all machines connected to the group, as shown in Figure 20.

Thus, the value of the logical clock of **Machine 2 (Windows - IP: 172.168.10.95)** be-

comes **(1)**, and of **Machine 1 (Linux - IP: 172.168.10.14)** and **Machine 3 (Windows - IP: 172.168.10.96)** becomes **(2)**, according to the *Lamport's* algorithm (Section 2.3).

Therefore, the Lamport's algorithm is executed as follows: **Machine 2 (Windows - IP: 172.168.10.95)** sends a message with its logical/physical date/time to the network (e.g., Logical Clock **(1)** and Physical Clock 08/28/1993 10:22). Consequently, **Machine 1 (Linux - IP: 172.168.10.14)** upon receiving the message will update its logical clock to **(2)**.

Consequently, **Machine 1 (Linux - IP: 172.168.10.14)** automatically triggers a response message to the group via **Send R:M/B/U...**, including its physical timestamp (e.g., `10/10/2004 20:48`). As a result of this transmission, **Machine 1** updates its logical clock to **(3)**.

Therefore, **Machine 2 (Windows - IP: 172.168.10. 95)** and **Machine 3 (Windows - IP: 172.168.10.96)** upon receiving the message returned by **Machine 1 (Linux - IP: 172.168.10.14)** will update their logical and physical clocks to **(4)** and 10/10/2004 20:48.

Consequently, after executing the *Lamport's* algorithm, machines 1, 2, and 3 will have the following logical/physical timestamps, respectively, **(3)/10/10/2004 20:48**, **(4)/10/10/2004 20:48** and **(4)/10/10/2004 20:48**, Figure 20[6].

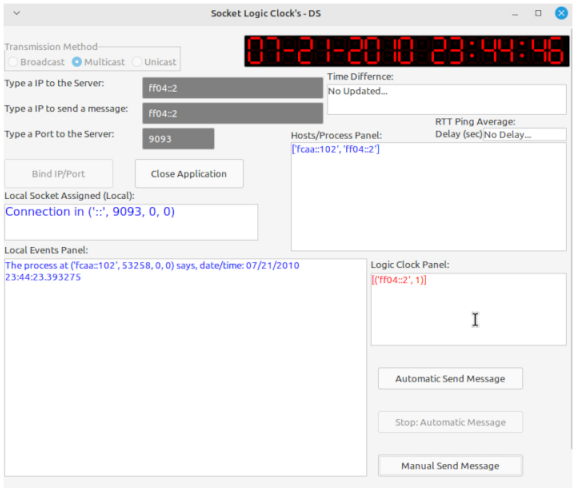
6.2.2 IPv6 Multicast

In this test, the settings are as follows:

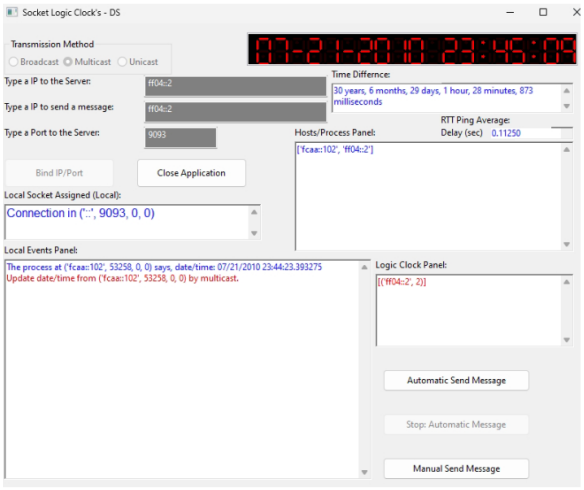
- **Machine 1:** Operating System Linux in Portuguese (PT-BR), Environment Physical, IPv6 fcaa::102/7 and Group IP ff04::2, Date and Time July 21/2010 at 11:40.
- **Machine 2:** Operating System Windows in English (EN-US), Environment Virtual, IPv6 fcaa::101/7 and Group IP ff04::2, Date and Time January 1/1980 at 10:14;
- **Machine 3:** Operating System Windows in English (EN-US), Environment Virtual, IPv6 fcaa::103/7 and Group IP ff04::2, Date and Time December 31/1990 at 12:08.

Similarly to *IPv4 Multicast*, the IPv6 Multicast test shows very similar results. However, the main difference is that the first machine to send the message was **Machine 1 (Linux - IPv6: fcaa::102)** as shown in Figure 21(a). In this context, the Lamport's algorithm is executed as follows: **Machine 1 (Linux - IPv6: fcaa::102)** sends a message with its logical/physical date/time to the network (e.g., Logical Clock **(1)** and Physical Clock **07/21/2010 23:44**). Consequently, **Machines 2 and 3 (Windows - IPv6's: fcaa::101 and fcaa::103)** upon receiving the message will update their logical and physical clocks to **(2)** and **07/21/2010 23:44**. It happens because Machines 2 and 3 have their date/time out of date relative to Machine 1 (e.g., **Machine 2: 01/01/1980 22:14** and **Machine 3: 12/31/1990 12:08**).

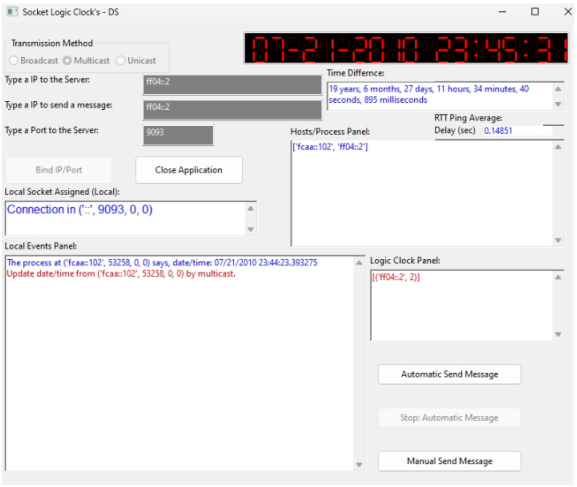
[6]The difference in seconds in the figures is due to the time taken to take the *print screen* on different machines.



(a) Executing LLCTT in a Linux Environment with IPv6 Multicast, Machine 1.



(b) Execution of LLCTT in Windows Environment with IPv6 Multicast, Machine 2.



(c) Execution of LLCTT in Windows Environment with IPv6 Multicast, Machine 3.

Figure 21: In (a) Machine 1, (e.g., Linux) is shown sending an IPv6 Multicast message to the network. In (b) and (c), this message is received by Windows machines 2 and 3; from then on, they update their logical and physical clocks.

Thus, after executing *Lamport*'s algorithm, machines 1, 2, and 3 will have the following logical/physical timestamps, respectively, **(1)/07/21/2010 23:44**, **(2)/07/21/2010 23:44** and **(2)/07/21/2010 23:44**, Figure 21[7].

6.3 Didactic Evaluation of the LLCTT

According to the methodology presented in Subsection 3.4, the result of the survey carried out via Google Forms is as follows:

[7]The difference in seconds in the figures is due to the time taken to take the *print screen* on different machines.

1. To the question “What do you think of the graphical interface?”, 76.5% of the students rated it as “Good”, 11.8% as “Excellent”, and 11.7% as “Basic”;
2. To the question “Does LLCTT generally facilitate learning about distributed systems?”, 76.5% answered “Yes” and 23.5% answered “Partially”;
3. To the question “Does the tool facilitate learning about logical clocks?”, 94.1% answered “Yes” and 5.9% answered “No”;
4. To the question “Does the process and logical clock panel help clarify Lamport’s algorithm?”, 94.1% of the students answered “Yes” and 5.9% answered “No”;
5. Finally, to the question “Which Distributed Systems concepts does LLCTT help you understand more clearly?”, the results are presented in Figure 22.

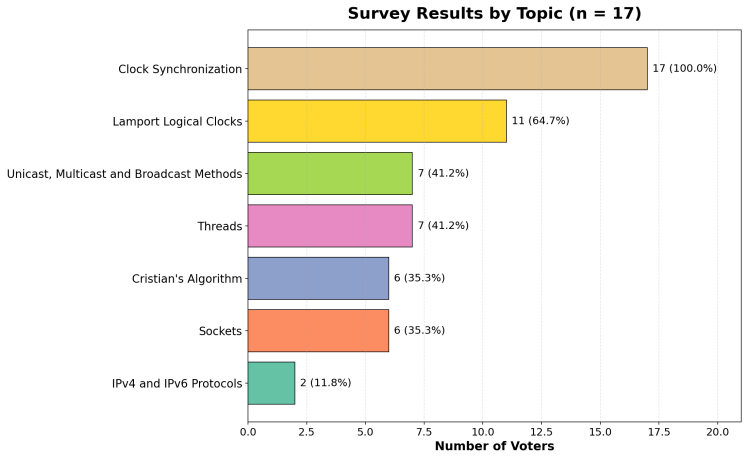


Figure 22: Survey Results for the Question “*Which Distributed Systems concepts does the LLCTT help you understand more clearly?*”.

As shown in Figure 22, when asked which distributed systems concepts the LLCTT helped them understand more clearly, clock synchronization was cited by all 17 respondents (100%). This was followed by Lamport’s logical clock algorithm, with 11 votes (64.7%).

These results reinforce that clock synchronization and Lamport’s logical clocks are inherently abstract and challenging concepts for students to grasp through traditional instruction alone. The fact that every student selected clock synchronization—and nearly two-thirds selected Lamport’s algorithm—suggests that the *LLCTT* is particularly effective as a pedagogical tool for these specific topics. By providing an interactive, visual environment, the *LLCTT* helps bridge the gap between theoretical complexity and practical understanding, especially for mechanisms that involve time management and logical ordering in distributed systems.

6.3.1 Didactic Use Script for LLCTT

The didactic use of *LLCTT* begins with a brief introduction to socket communication, clock synchronization, and Lamport logical clocks. Students then execute the tool and explore its graphical

interface, including transmission method selection, IP/port configuration, the local events panel, and the logical clock panel.

Next, the instructor configures a communication mode (*Broadcast*, *Multicast*, or *Unicast*) and starts the system. Students send messages manually or automatically and observe how communication events update the event log and logical clock values. This practical interaction helps illustrate event ordering, synchronization, and causality in distributed systems.

Finally, the observed behavior is discussed and related to theoretical concepts. A complete installation and usage manual for *LLCTT* is available in the project's *GitHub* repository.

7 Conclusion and Future Work

This paper presented a novel didactic tool for teaching clock synchronization in distributed systems, the *Lamport Logical Clock Teaching Tool (LLCTT)*. To validate the proposed solution, a systematic literature review was conducted to evaluate existing educational software. The review highlighted that, while several tools provide valuable abstract simulations, *LLCTT* distinguishes itself through its architectural choice to bridge the gap between local, memory-based simulators and real network communication.

Furthermore, we evaluated the proposal in a network laboratory through a series of practical tests. For instance, we synchronized different computers with varying initial timestamps, combining nodes running *Linux* and *Windows*. The results demonstrated the successful alignment of the physical clocks followed by the causal execution of the logical clocks. In this context, students can visualize, in detail and independently, both the physical time adjustments and the logical increments.

Among the main features and contributions of *LLCTT*, the following stand out:

1. A user-friendly and interactive graphical interface developed using *WX-Python*;
2. Native support for both *IPv4* and *IPv6* network protocols;
3. Compatibility with multiple transmission methods, including *Unicast*, *Broadcast*, and *Multicast*;
4. Cross-platform installation support for both *Linux* and *Windows* operating systems;
5. Implementation based on key distributed systems concepts, such as *Datagram Sockets*, *Multithreading*, and Object-Oriented Programming;
6. A proprietary licensing model, available through commercial licensing plans and distributed via GitHub;
7. Comprehensive documentation and installation instructions available on GitHub for both operating systems[8];

[8] https://github.com/dioxfile/Logic_Clock_Didatic_Tool.

8. A dual-layered didactic approach that first mitigates a real-world infrastructure problem (desynchronization caused by depleted *CMOS* batteries) via a physical highest-timestamp heuristic, thereby establishing a reliable environment to practically demonstrate Lamport's logical clocks (Lamport, 1978) without theoretical confusion.

Ultimately, *LLCTT* enables hands-on, lab-based activities that serve as highly effective didactic exercises in distributed systems courses. Through its use in practical classes, students were able to experience and test theoretical concepts—particularly causal ordering—over a tangible, distributed infrastructure, which represents one of the tool's most significant contributions to the learning process. As future work, we plan to extend *LLCTT* to support the simulation and visualization of Vector Clocks.

Acknowledgments

The authors would like to express their sincere gratitude to the State University of Mato Grosso (UNEMAT) for providing the development environment and institutional support necessary for this work.

The authors also acknowledge AGINOV, the Innovation Agency of UNEMAT, for fostering innovation, research, and entrepreneurship, as well as for its support in the registration of LLCTT at the Brazilian National Institute of Industrial Property (INPI).

Authors' Contributions

The authors contributed as follows:

Author 1 (Diógenes Antonio Marques José) developed *LLCTT*, managed the project's *GitHub* repository, and contributed to the writing and revision of the manuscript;

Author 2 (Bruno Hernandes Carrilho Martins) adapted *LLCTT* for *Windows* compatibility and performed software testing and validation.

Competing interests

The authors declare that they have no competing interests.

References

Adams, C., Cain, P., Pinkas, D., & Zuccherato, R. (2001). Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP) [Accessed on: January 20, 2024]. http://www.ietf.org/rfc/rfc3161.txt [GS Search].

Antunes, J. (2014). Relógios de Lamport [Accessed on: June 12, 2023]. https://github.com/joao8tunes/RelogiosLamport/tree/master/

Barcellos, C. (2023). Simulador de relógios lógicos de Lamport [Accessed on: January 23, 2024]. https://github.com/camilafbarcellos/LamportClockSim/

Cândido, P. H. V., Morais, J. V. F., Santos, S. M., Zevarez, V. A. O., Cavalherie, L. F. A., Gois, P. H. M., & Silva-Santos, C. H. (2021). Panda: Uma nova ferramenta web responsiva para auxiliar no ensino e comunicação de pessoas com limitações psicomotoras. *Brazilian Journal of Computers in Education (RBIE)*, *29*, 25–47. https://doi.org/10.5753/RBIE.2021.29.0.25 [GS Search].

Cristian, F. (1989). Probabilistic clock synchronization. *Distributed computing*, *3*, 146–158. https://doi.org/10.1007/BF01784024 [GS Search].

Gusella, R., & Zatti, S. (1989). The accuracy of the clock synchronization achieved by TEMPO in Berkeley UNIX 4.3BSD. *IEEE Transactions on Software Engineering*, *15*(7), 847–853. https://doi.org/10.1109/32.29484 [GS Search].

Lamport, L. (1978). Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, *21*(7), 558–565. https://doi.org/10.1145/359545.359563 [GS Search].

Lamport, L., & Melliar-Smith, P. M. (1985). Synchronizing clocks in the presence of faults. *Journal of the Association for Computing Machinery*, *32*(1), 27. https://doi.org/10.1145/2455.2457 [GS Search].

Luzia, S. I. (2004). Um ambiente para ensino de algoritmos de consenso [Master Tesis in Science Computing]. https://doi.org/10.47749/T/UNICAMP.2004.363296 [GS Search].

Mills, D. L. (1989). Network Time Protocol version 1 [Accessed on: January 19, 2024]. https://tools.ietf.org/html/rfc1059 [GS Search].

Rodrigues, R. J. S., Lima, R. A., & José, D. A. M. (2017). Algoritmo para sincronização de relógios físicos em sistemas distribuídos [Escola Regional de Redes de Computadores (ERRC) 2017 (UFSM)]. https://www.ufsm.br/app/uploads/sites/360/2019/06/ANAIS_ERRC_2017.pdf [GS Search].

Santana, T. (2017). Simulação do algoritmo de Lamport em Java [Accessed on: January 10, 2024]. https://github.com/terciosantana/lamport/

Silva, D. L. (2020). Relógio lógico de Lamport [Accessed on: February 12, 2024]. https://acervolima.com/relogio-logico-de-lamport/

Tanenbaum, A. S., & Steen, M. V. (2023). *Distributed systems* (4th). Pearson Education. https://www.distributed-systems.net/index.php/books/ds4/ds4-ebook/ [GS Search].

Taylor, W., & Stewart, J. (2016). Lamport and vector logical clocks. https://github.com/fire-at-will/Logical-Clock/.

Tedesco, L. (2019). Sincronização de relógios com o algoritmo de Berkeley e Java RMI [Accessed on: December 13, 2023]. https://github.com/lucianetedesco/algoritmo-berkeley/

Tran, T. V. (2018). Lamport clock implementation [Accessed on: December 20, 2023]. https://github.com/tuvtran/LamportClock/

Whittaker, M. (2015). Lamport's logical clocks [Accessed on: January 07, 2024]. https://mwhittaker.github.io/blog/lamports_logical_clocks/