Sistemas Distribuídos

Synchronization

Eurico Pedrosa

António Rui Borges

Universidade de Aveiro - DETI

2025-04-02

Summary

- Time concepts
  - Global time
  - Local time
  - Logical time
- Adjustment of local time
  - Problem characterization
  - Cristian method
  - Berkeley algorithm
  - Network Time Protocol
- Suggested reading

Time

Concepts

Physical phenomena—especially human activity—occur in both space and time. Among these dimensions, **time** plays a critical role in characterizing events, determining their order, and assessing potential causal relationships between them.

In distributed systems, the concept of time can be approached from several perspectives, depending on how the observer perceives reality:

- **global time** – Time as perceived by an external or universal observer.
- **local time** – Time as perceived individually by each entity being observed.
- **logical time** – Time derived from the flow of information, used to order events based on causality rather than absolute timestamps.

Global Time

Time cannot be directly measured. Instead, we infer its passage by observing the **periodic motion of well-defined physical objects**, leveraging the intrinsic link between **time and space**.

One classical approach is **Astronomical time**.

This system is based on the periodic movements of celestial bodies, primarily:

- The **Earth's orbit around the Sun**, where each complete revolution defines a **solar year**.
- The **Earth's rotation on its axis**, where each complete spin defines a **solar day**.

Global Time

Each solar day is subdivided into:

- **24 hours**
- **1 hour** = 60 **minutes**
- **1 minute** = 60 **seconds**

Hence, the **solar second**—the standard unit of time—is defined as **1/86,400 of a solar day**. A **solar year** is approximately **365 days and 6 hours**.

Global Time - Atomic Time

The **periodic motion of celestial bodies**—while useful—is not sufficiently stable to serve as a precise standard over very long periods.

Scientific studies over recent decades have shown that:

- The **Earth's rotation** gradually **slows down** due to **tidal friction** and **atmospheric drag**.
- It also undergoes minor **irregular oscillations** in angular speed, likely caused by **turbulence in the Earth's core**.

Due to these limitations, the definition of the **second** was re-evaluated with the introduction of **atomic clocks**, which offer extremely stable and consistent timekeeping.

Global Time - Atomic Time

Current Definition of the Second

The **second** is defined as the time it takes for **9,192,631,770 periods** of the radiation corresponding to the **transition between two hyperfine levels** of the ground state of the **cesium-133 atom**, **at rest** and at a **temperature of 0 Kelvin**

Global Time - TAI

International Atomic Time (TAI) is a globally accepted standard for precise timekeeping. It is computed as the **weighted average of time readings from over 200 atomic clocks** maintained by national metrology institutes across the world.

This process is coordinated by the **International Bureau of Weights and Measures (BIPM)**, specifically through its **International Bureau of the Hour (BIH)**.

Global Time - TAI

Key Facts

- The duration of the **standard second** was defined to match the **solar second** on **January 1, 1958**, ensuring continuity with astronomical time.
- On **January 1, 1977**, a **relativistic correction** was applied to account for **time dilation** effects caused by variations in **Earth's gravitational field**—since atomic clocks are located at **different altitudes**, they tick at slightly different rates due to general relativity.

Global Time - UTC

Coordinated Universal Time (UTC) is the principal time standard used for regulating clocks and timekeeping in human activities. It is **based on International Atomic Time (TAI)** and is expressed in **standard seconds**.

To stay in alignment with **astronomical time**, UTC occasionally adjusts for the **slowing and irregularities of the Earth's rotation**. This is done by **adding or (in theory) subtracting a leap second**—though subtraction has not occurred to date.

Global Time - UTC

Key Characteristics:

- The **second and its submultiples** (milliseconds, microseconds, etc.) are always **constant**.
- However, **larger units** (minutes, hours, days) may vary slightly over time due to leap second adjustments.
- **Leap seconds** are typically inserted at the end of **June or December**.

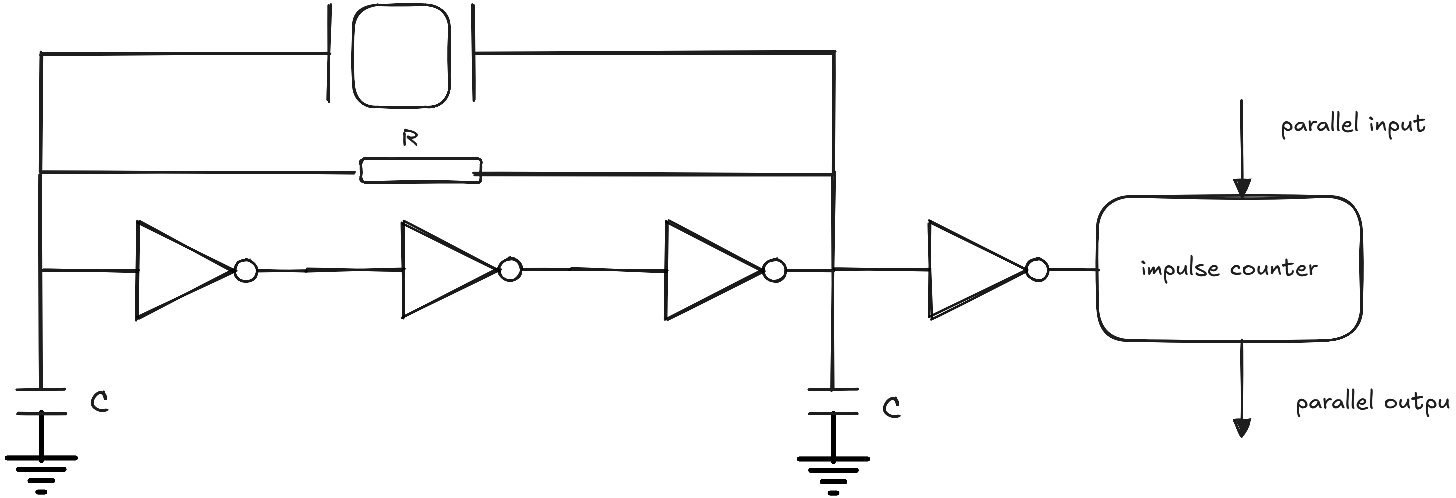
Sources of UTC Time (with uncertainty levels):

- **Shortwave radio transmitters**: ±10 ms
- **Geostationary satellite systems** (e.g., GEOS, GPS): ±0.5 ms
- **Internet time servers** using **NTP** (Network Time Protocol): ±50 ms

Local Time

A **computer system's clock** consists of two main components:

- An **oscillator circuit**, typically driven by a **quartz crystal**, which produces periodic electrical impulses.
- An **impulse counter**, which increments a stored value each time an impulse is received.



Local Time

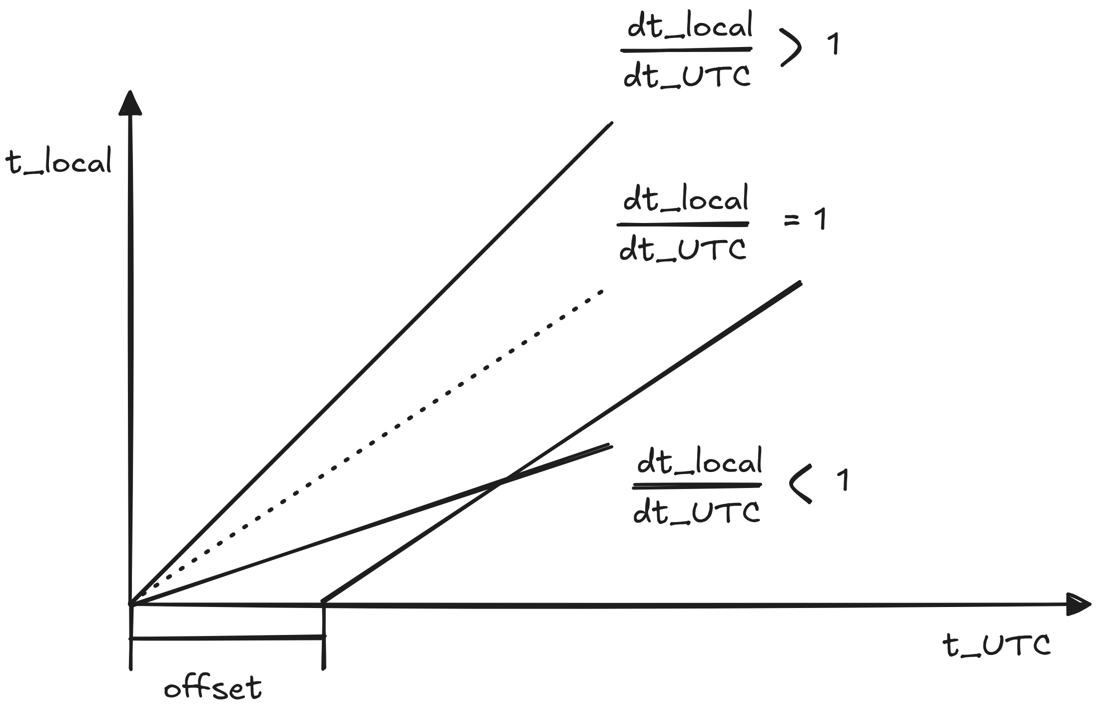
At any given moment, the **current count** can be:

- **Read** via the **parallel output**, and
- **Set** to a specific value via the **parallel input**.

This **counting value** can be **converted to real time** if a **time origin** is established. For example, in **Unix-based systems**, the origin is set to **00:00:00 on January 1, 1970** (commonly referred to as the Unix epoch).

Local Time

A computer clock may **display incorrect time** due to two main sources of error: **drift** and **offset**.



Local Time

1. **Offset**
   - The clock's count differs from the correct value by a **fixed number of impulses**.
   - This typically results from an **incorrectly defined time origin**.
2. **Drift**
   - The **oscillator's frequency deviates** from its nominal (ideal) value.
   - This deviation is often caused by **environmental factors** such as **temperature** and **humidity**, leading to a gradually increasing error in the **counting rate**.

For example, a **quartz-controlled oscillator** typically exhibits a **drift** on the order of **1 part in 10^6 per second**.

Local Time Adjustments

Problem Characterization

Synchronizing the **local clocks** of computer systems (nodes) in a distributed or parallel environment is essential to ensure **temporal consistency**. This synchronization can be addressed in two distinct ways:

1. External Synchronization

- Given a **trusted UTC source** $S(t)$ and a **maximum allowable deviation** Δ, ensure that each local clock $C_{ki}(t)$ (for node $i = 0, 1, ..., N - 1$ satisfies:

$$\forall i \in \{0, 1, ..., N - 1\}, \quad |S(t) - C_{ki}(t)| < \Delta$$

2. Internal Synchronization

- Even if there is **no access to an external time source**, ensure that the **difference between any two local clocks** remains within the allowed uncertainty Δ:

$$\forall i, j \in \{0, 1, ..., N - 1\}, \quad |C_{ki}(t) - C_{kj}(t)| < \Delta$$

Problem Characterization

In distributed or parallel systems, it is assumed that:

- The **processing nodes** are connected by a specific **interconnection topology**.
- Communication between nodes occurs via **message passing**.
- Messages have a **finite transmission time**, but **no known upper bound** can be guaranteed. This implies the following property of the system:

  $$\forall L \in R^+, \quad \exists t_M \text{ such that } t_M > L$$

  Where t_M is the **transmission time** of a message.
- This means **no matter how large a delay L is**, it's always possible that a message may take **longer than L** to arrive.

Such uncertainty makes **precise timing coordination** in distributed systems more challenging.

Problem Characterization

Clock Adjustment and Monotonicity

When synchronizing clocks in distributed systems, **time adjustments must always preserve the monotonicity of local time**.

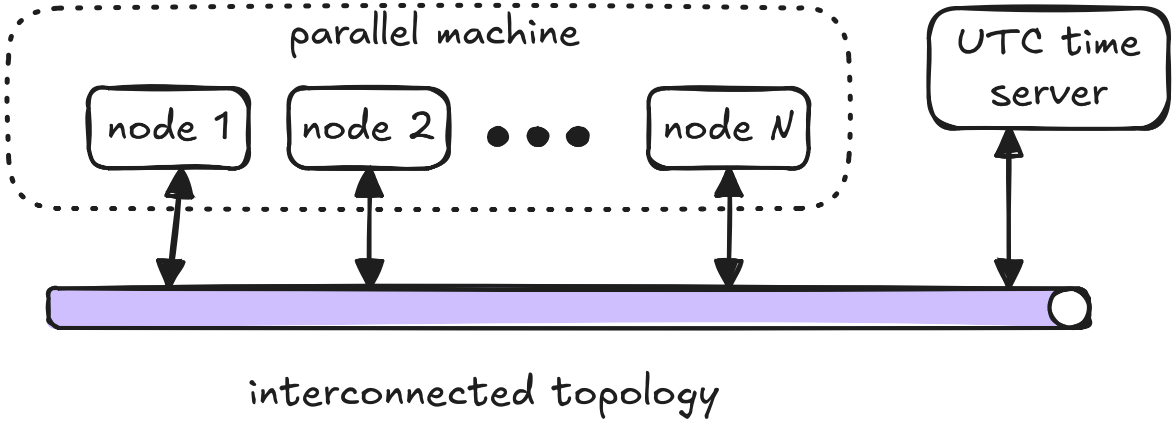
That is, the local clock should **never move backward**, only forward or pause.

Why is monotonicity important?

- **Preserving causality**: If a clock moves backward, it may appear that an event occurred **before** its cause, violating logical causality.
- **Correct event ordering**: Time-based logs, timestamps, and message ordering rely on non-decreasing time to remain consistent.
- **System stability**: Many algorithms assume time is always progressing; non-monotonic clocks can lead to errors or unpredictable behavior in scheduling, timeout handling, and replication protocols.

Cristian's Method

Cristian's Method is an **external synchronization technique** used in distributed systems. It assumes the availability of a **UTC time server**.



Cristian's Method

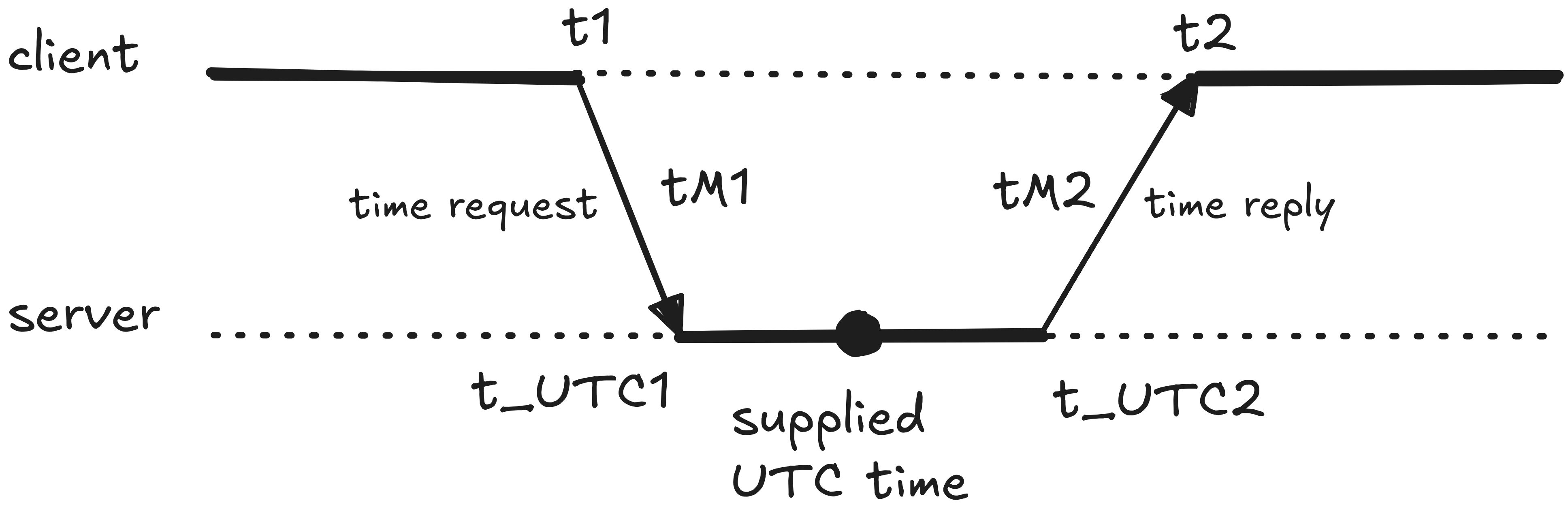
How it works

- **Proactively**, each client node requests the current time by sending a message to the **UTC server**.
- Upon receiving the request, the **server responds with the current time** in a **predefined format**.
- The client then adjusts its **local clock**, accounting for **message transmission delays**, to align it as closely as possible with the **UTC time**.

This method relies on the assumption that:

- The message **round-trip time (RTT)** is symmetric and small,
- And the system clock adjusts **monotonically** (never moves backward).

Cristian's Method



Cristian's Method - Decomposition

Step-by-step synchronization process:

1. At **local time** t_1, the **client sends a request** to the UTC server.
   - The **message transmission time** is t_{M1}.
   - The request is received by the server at **UTC time** $t_{\mathrm{UTC}\ 1}$.
2. The **server replies** with an estimated **UTC time** t_{UTC}, adjusted to match approximately the **middle of the server's processing interval**.
3. The **reply is sent** at **UTC time** $t_{\mathrm{UTC}\ 2}$.
   - It reaches the client at **local time** t_2 after a message delay of t_{M2}.

Cristian's Method - Decomposition

Clock Adjustment by the Client

- The client has access to:
  - t_1, t_2: local times when the request and response were observed
  - t_{UTC}: estimated time from the server
- Assumptions:
  - The **local clock drift** over the short interval $t_2 - t_1$ is negligible.
  - The **server processing time** is negligible compared to message transmission time.
- Estimated Offset:

$$\text{offset} = t_{\mathrm{UTC}} - \frac{t_1 + t_2}{2}$$

Cristian's Method - Decomposition

- Estimated Uncertainty:
  - Assuming one message experienced **minimum delay** (t_{MIN}), the **worst-case uncertainty** is:

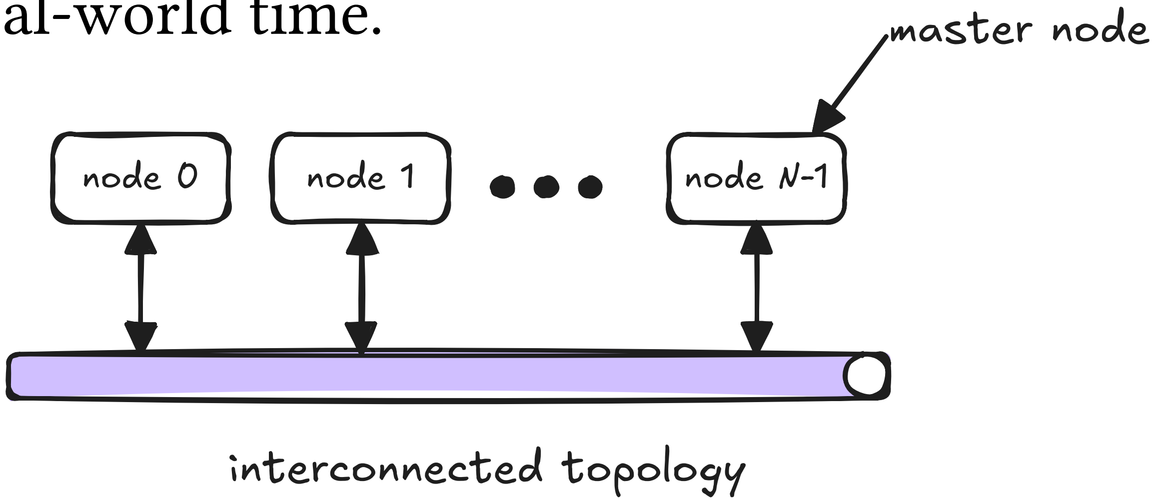
$$\Delta_{\text{est}} = \frac{t_2 - t_1}{2} - t_{\text{MIN}}$$

- Robustness
  - If the estimated uncertainty Δ_{est} exceeds the **accepted threshold** (e.g., due to network load or server delay), the **client should discard the result and retry later**.

Berkeley algorithm

The **Berkeley Algorithm** is an **internal clock synchronization method** used when **no UTC time source** is available.

Its objective is to ensure that all **processing nodes in a distributed system** maintain **synchronized local clocks**, even if they are not aligned with real-world time.



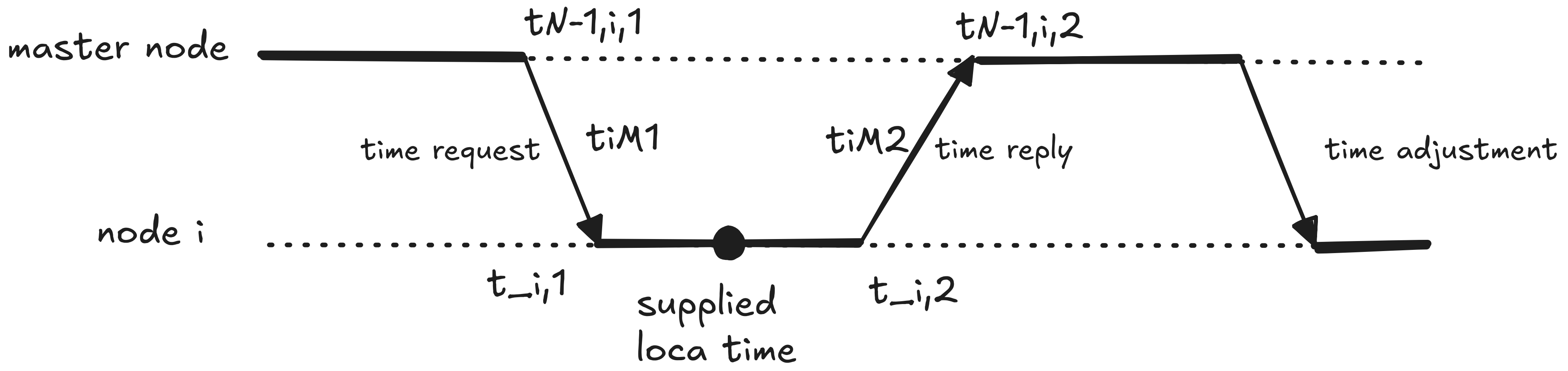
Berkeley algorithm

How it works:

- Periodically, one node is elected or designated as the **master node**.
- The master **proactively polls** all nodes in the system—including itself—asking for their **current local time**.
- After collecting all responses, the master:
  1. **Calculates the average time offset** between itself and the other nodes.
  2. **Computes the correction** needed for each node to align with the average.
  3. **Sends back the offset corrections** to each node, including itself.

Each node then **adjusts its clock** based on the received correction, maintaining **internal synchronization** across the system.

Berkeley algorithm



Berkeley algorithm - Decomposition

Step-by-step Process

1. The **master node** initiates synchronization by sending a request to every other node (including itself) at **local time** $t_{N-1,i,1}$.
   - Each message has a **transmission time** $t_{\{i,M1\}}$.
   - Node (i) receives the request at **local time** $t_{i,1}$, for $i = 0, 1, …, N - 1$.

2. Each node responds with a timestamp t_i, which is adjusted to reflect the **middle of the time window** it used to process the request.

3. The reply is sent at **local time** $t_{i,2}$, transmitted back to the master with delay $t_{i,M2}$.
   - The master receives each reply at **time** $t_{N-1,i,2}$.

Berkeley algorithm - Decomposition

4. Once all replies are collected, the **master node**:
   - **Estimates clock offsets** by comparing the received values to its own clock.
   - **Computes a global average deviation** (excluding outliers if necessary).
   - Sends a message to **each node** with the **correction** it should apply to adjust its local clock.

This ensures that **all clocks in the system are synchronized** as closely as possible to a common internal time, even in the absence of an external UTC source.

Berkeley algorithm - Decomposition

Offset Estimation by the Master Node

The **master node** uses the following timestamps to estimate each node's clock offset and uncertainty:

- $t_{N-1,i,1}$: Time the request was sent to node i
- $t_{N-1,i,2}$: Time the reply was received from node i
- t_i: Timestamp reported by node i, adjusted to the middle of its processing interval

Assumption:

- The **drift** of the master's clock during the interval $t_{N-1,i,2} - t_{N-1,i,1}$ is **negligible**.

Berkeley algorithm - Decomposition

Offset Estimation (using Cristian's formula):

$$\text{offset}(i) = t_i - \frac{t_{N-1,i,1} + t_{N-1,i,2}}{2}$$

Uncertainty Estimation:

$$\Delta_{\text{est}(i)} = \frac{t_{N-1,i,2} - t_{N-1,i,1}}{2} - t_{\text{MIN}}$$

for all $i = 0, 1, ..., N - 1$

Berkeley algorithm - Decomposition

Averaging and Correction:

- The **master node** computes the **average offset**, $\text{offset}_{\text{med}}$, **excluding nodes** whose uncertainty $\Delta_{\text{est}(i)}$ exceeds the **acceptable threshold**.
- Then it sends to each node the **adjustment to apply**:

$$\text{adjustment}(i) = \text{offset}_{\text{med}} - \text{offset}(i)$$

Key Remark:

- Since this adjustment is a **differential value**, the **transmission delay of this final message does not introduce additional uncertainty**.

Network Time Protocol

Network Time Protocol

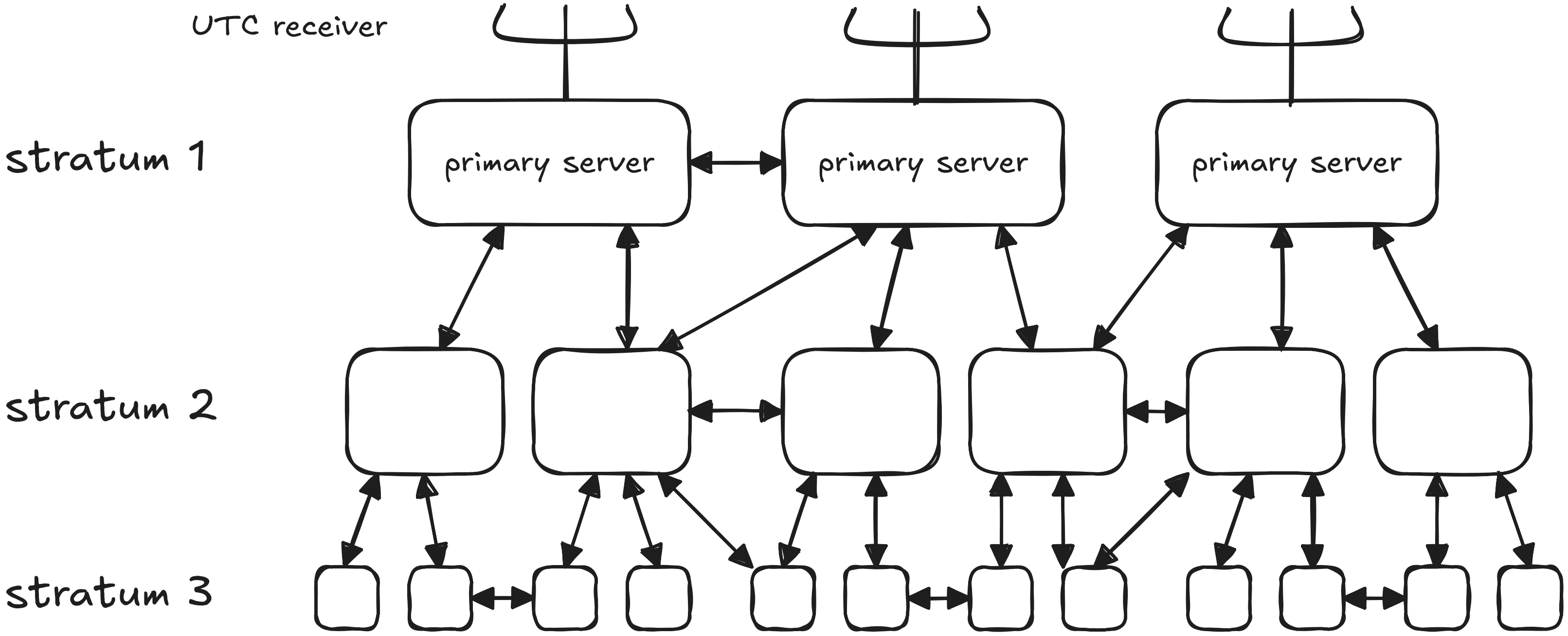
While previously discussed methods focus on **synchronizing clocks within local area networks**, the **Network Time Protocol (NTP)** is designed specifically for **global synchronization over the Internet**.

Goals of NTP:

- **Accurate synchronization**: Enable any computer system connected to the Internet to **adjust its local clock** with **reasonable precision**.
- **Timely adjustments**: Perform clock corrections at a **sufficiently frequent rate** to prevent **noticeable drift** over time.
- **Resilience to disconnection**: Ensure the service remains **reliable and available**, even in the face of **temporary connectivity loss** with specific time servers.
- **Security and robustness**: Provide **protection against malicious interference** or corrupted time data, helping preserve **integrity and trust** in time synchronization.

Network Time Protocol

NTP relies on a **hierarchical structure** of time servers, organized into multiple **levels called strata**, to distribute accurate time across the Internet.



Network Time Protocol

Hierarchical Organization:

- **Stratum 1**: These are **primary servers** that are **directly connected to a UTC source** (e.g., atomic clocks or GPS receivers). They serve as the **root** of the synchronization hierarchy.
- **Stratum 2 and below**: These are **secondary servers**. Each server in **stratum n** synchronizes its clock with one or more servers in **stratum $n-1$**.
- **Lateral coordination**: Servers in the **same stratum** can also **synchronize with each other**, improving **robustness** and **stability** of the time information across the network.

This structure enables scalable, fault-tolerant, and resilient time distribution throughout the Internet.

Network Time Protocol

Accuracy and Stratum Depth

- As one moves **down the hierarchy** (from **Stratum 1** to higher-numbered strata), the **uncertainty of time information increases**.
- This is due to the **cumulative effect of synchronization errors** introduced at each stage in the hierarchy.

Dynamic Reconfiguration

NTP maintains **resilience and availability** through **dynamic restructuring** of the synchronization sub-tree:

- If a **primary server (Stratum 1) loses access to its UTC source**, it **demotes itself to Stratum 2**, acting as a **secondary server**.
- If a **time source becomes unavailable** to a server at any level, it will **search for an alternative server**—either in the **same stratum** or a **higher one**—to maintain synchronization.

This dynamic behavior ensures **fault tolerance** and **continuous service** even under partial network failure or loss of connectivity.

Network Time Protocol

In the **Network Time Protocol (NTP)**, **pairs of nodes** exchange **synchronization messages continuously**:

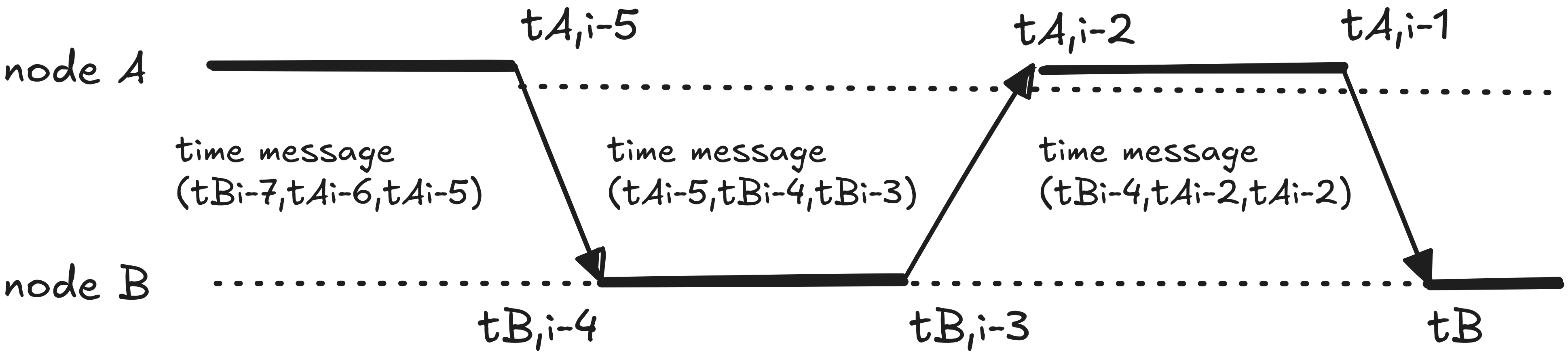
- If node **B** is in a **lower stratum** than node **A**, it uses the exchange to **adjust its local clock**.
- If both nodes are in the **same stratum**, they perform a **mutual adjustment** for increased accuracy and robustness.

Message Structure

Each synchronization exchange includes **four timestamps**:

1. $t_{B,i-3}$: Local time at **B** when the **last message was received**.
2. $t_{A,i-2}$: Local time at **A** when the **last message was received**.
3. $t_{A,i-1}$: Local time at **A** when the **current message is sent**.
4. $t_{B,i}$: Local time at **B** when the **current message is received**.

Network Time Protocol



Network Time Protocol

Purpose

These four timestamps are used by node **B** to:

- **Estimate the offset** between clocks A and B:

$$\text{offset} \approx \frac{\left(t_{B,i-3} - t_{A,i-2}\right) + \left(t_{B,i} - t_{A,i-1}\right)}{2}$$

- **Estimate the uncertainty (network delay asymmetry)** using round-trip delay and timing variation.

This method enables **precise synchronization** even across **unpredictable network links**.

Network Time Protocol

In a time synchronization exchange between two nodes, **B** and **A**, node **B** makes the following assumptions and calculations to estimate the clock offset and uncertainty.

Assumptions:

- The **drift** of both local clocks is **negligible** during the short intervals:
  - $t_{B,i} - t_{B,i-3}$
  - $t_{A,i-1} - t_{A,i-2}$

Offset Estimation

Let the **offset** between the two clocks be:

$$\text{offset} = Ck_{A(t)} - Ck_{B(t)}$$

It is estimated as:

$$\text{offset}_{\text{est}} = \frac{t_{A,i-1} + t_{A,i-2}}{2} - \frac{t_{B,i} + t_{B,i-3}}{2}$$

Uncertainty Estimation

Assuming **one of the two message transmissions** experienced **minimum delay** t_{MIN}, the **worst-case uncertainty** is estimated as:

$$\Delta_{\text{est}} = \frac{(t_{A,i-2} - t_{B,i-3}) + (t_{B,i} - t_{A,i-1})}{2} - t_{\text{MIN}}$$

Statistical Filtering

- Node **B** maintains a **stream of offset/uncertainty pairs** $\text{offset}_{\text{est}}, \Delta_{\text{est}}$.
- These values are **filtered statistically** to compute a **refined and stable final offset**.
- The process is typically **repeated with multiple servers**.
  - The results are compared.
  - If discrepancies or inconsistencies arise, **a change of synchronization server(s)** may be triggered.

Suggested Reading

- M. van Steen and A.S. Tanenbaum, Distributed Systems, 4th ed., distributed-systems.net, 2023.