

# Time in Distributed Systems

**Sirio Scipioni, Roberto Baldoni**

Sapienza University of Rome

[scipioni@dis.uniroma1.it](mailto:scipioni@dis.uniroma1.it)

<http://www.dis.uniroma1.it/~scipioni>

MIDLAB - <http://www.dis.uniroma1.it/~midlab>

# Introduction

## In a Distributed System

1. Processes run on different nodes interconnected by mean of a network (LAN or WAN)
2. Processes cooperate to complete a computation
3. Processes communicate exclusively through Message-based communications

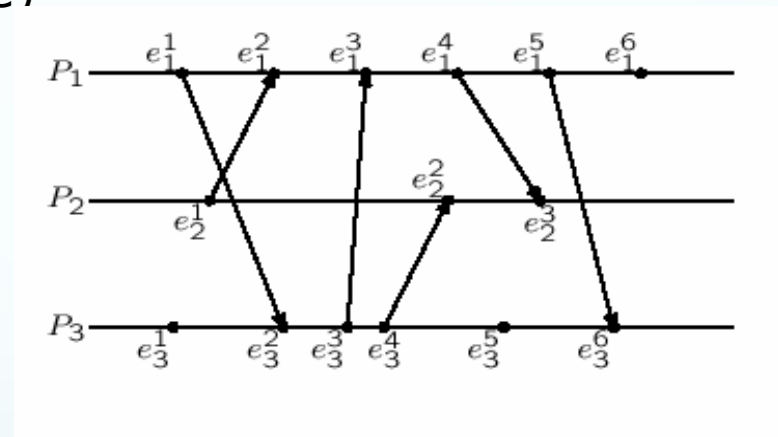
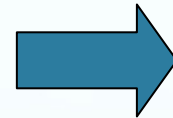
### **A crucial remark:**

- Several algorithms require an order among events in order to terminate or to correctly return a result
  - Aviation Traffic Control
  - Network monitoring, measurement and control
  - Stock market buy and sell orders, etc...

# System Model

- System:  $n$  processes and communication channels
- Each process produces a sequence of events
  - Internal and External (send/receive)
  - $e_i^k$ ,  $k$ -th event produced by  $p_i$

Evolution of a computation can be represented by a *space-time diagram*



- Time is crucial in distributed system to order subsequent events
  - E.g. Different PCs have to ordinate consistently several transactions relatively to an e-commerce website.

# History of Computations

We denote with  $\rightarrow_i$  the ordering relationship between two events in the same processes  $p_i$ :

$e \rightarrow_i e'$  if and only if  $e$  is happened before  $e'$  in  $p_i$

## Local History

Sequence of events produced by a process

$$\text{history}(p_1) = \mathbf{h}_1 = \langle \mathbf{e}_1^1, \mathbf{e}_1^2, \mathbf{e}_1^3, \mathbf{e}_1^4, \mathbf{e}_1^5, \mathbf{e}_1^6 \rangle$$

## Partial Local History

Prefix of local history

$$\mathbf{h}_1^m = \mathbf{e}_1^1 \dots \mathbf{e}_1^m$$

## Global History:

Set containing every local history

$$\mathbf{H} = \cup_i \mathbf{h}_i \text{ per } 1 \leq i \leq n$$

# Time in Distributed Systems (1/2)

## **Timestamping**

Each PC attaches a label to each event (using a timestamp). In this way it should be possible to realize a global history of the system.

## **“Naïf” solution:**

Each process timestamps events by mean of its physical clock

# Time in Distributed Systems (1/2)

## But... Is Timestamping really a solution?

- It is possible to define an order among events produced by the same process!!!
- But what's happen when we consider several distinct processes running on different PCs?

In a distributed system in presence of network delay, processing delay, etc... is **impossible** to realize a common clock shared among every process.

# Time in Distributed Systems (2/2)

**But through timestamping we can :**

Try to synchronize, with some *approximations*, physical clocks accessed by each process through appropriate algorithms.

In this case a process can label events using its physical local clock (synchronized within an approximation called *synchronization accuracy* or *synchronization error*).

# Time in Distributed Systems (2/2)

Timestamping is based on the notion of physical time (***physical clock***)

*In asynchronous model it is not feasible maintaining processes' clocks synchronized with a precision greater than a predefined bound in each time interval.*

A completely different approach is based on the logical time (***logical clock***).

# Physical Computer Clocks

Application processes access to a local clock obtained by operating system reading a local hardware clock.

- Hardware clocks consist of an oscillator and a counting register that is incremented at every tick of the oscillator.

At real time  $t$ , the operating system reads the hardware clock  $H_i(t)$ , therefore it produces the software clock

$$C_i(t) = \alpha H_i(t) + \beta$$

□

The software clock  $C_i(t)$  is obtained by scaling and adding an offset to hardware clock.

# Physical Computer Clocks

$C_i(t)$  approximates the physical time  $t$  at processing  $p_i$ .  $C_i(t)$  may be implemented by a 64-bit word, representing nanoseconds that have elapsed at time  $t$ .

- Generally this clock is not completely accurate: it can be different from  $t$ .
- $C_i$  can be used such as timestamp for event produced by  $p_i$ .

But... How much should be smaller the granularity (**resolution**) of software clocks (the time interval between two consecutive increments of software clock) to distinguish between two different events?

$$T_{\text{resolution}} < \Delta T \text{ between two notable events}$$

# Physical Clocks in Distributed Systems

- Different local clocks can have different values:
  - **Skew:** “the difference in time between two clocks”  $|C_i(t) - C_j(t)|$  (*Galli*)
  - **Drift:** clocks update their counting register with different frequencies (phenomenon is due to difference in material, temperature variation, etc...), then they deviate.
  - **Drift Rate:** “the gradual misalignment of once synchronized clocks caused by the slight inaccuracies of the time-keeping mechanisms” (difference for time unit as regards to an ideal clock), e.g. drift rate of 2microsec/sec means clock increases its value of 1sec+2microsec for each second.
    - Ordinary quartz clocks deviate nearly by 1 sec in 11-12 days. ( $10^{-6}$  secs/sec).
    - High-precision quartz clock drift rate is  $10^{-7}$ - $10^{-8}$  secs/sec

# Universal Time Coordinated (UTC)

- UTC is an international standard: the base of any international measure.
- Based on International Atomic Time (TAI): 1 sec = time a cesium atom needs for 9192631770 state transitions.
  - The TAI-day is about 3 msec shorter than a astronomical day consequently the Bureau International de l'Heure inserts 1 sec, if the difference between a day and a TAI-day is more than 800 msec
  - Physical clocks based on atomic oscillators are the most accurate clocks (drift rate  $10^{-13}$ )

# Universal Time Coordinated (UTC)

- UTC-signals come from shortwave radio broadcasting stations or from satellites (GPS) with an accuracy of
  - 1 msec for broadcasting stations
  - 1  $\mu$  sec for GPS
  - $\gg$  1 msec for UTC available via phone line
- Receivers of UTC-signals can be connected to Computers and can be used to synchronize local clocks with the real time

# Internal/External Synchronization

- **External Synchronization**

- Synchronization of process' clocks  $C_i$  with an authoritative external source  $S$
- Let  $D > 0$  be the synchronization bound and  $S$  be the source of UTC

- Clocks  $C_i$  (for  $i = 1, 2, \dots, N$ ) are ***externally synchronized*** with a time source  $S$  (UTC) if for each time interval  $I$ :

$$|S(t) - C_i(t)| < D \text{ for } i = 1, 2, \dots, N \text{ and for real time } t \text{ in } I$$

- We say that clocks  $C_i$  are ***accurate*** within the bound of  $D$

# Internal/External Synchronization

- **Internal Synchronization**

- Synchronization of process' clocks  $C_i$  with each other
- Let  $D > 0$  be the synchronization bound and  $C_i$  and  $C_j$  are clocks at processes  $p_i$  and  $p_j$  respectively

- Clocks are **internally synchronized** in a time interval  $I$ :  
 $|C_i(t) - C_j(t)| < D$  for  $i, j = 1, 2, \dots, N$  and for all time  $t$  in  $I$

- We say that clocks  $C_i, C_j$  **agree** within the bound of  $D$

# Physical Clock Synchronization

## Notes:

- Note that clocks that are internally synchronized are not necessarily externally synchronized. i.e. even though they agree with each other, they drift collectively from the external time source.
- On the other hand, a set of processes  $P$ , externally synchronized within the bound of  $D$ , is also internally synchronized within the bound of  $2D$ .
  - This property directly follows from the definition of internal and external clock synchronization.

# Physical Clock Synchronization

	Properties		Requirements	
Authors	Algorithm Paradigm	Synchronization Paradigm	Communication Topology	Assumptions on message delays
Lamport	Deterministic	Internal	Clique	Known bound
Cristian	Probabilistic	Internal	Star (Master-Slave)	Known distribution
Mills (NTP)	Probabilistic	External	Hierarchical (Master-Slave)	Known distribution
D. Dolev et al.	Deterministic	HeartBeat	Clique	Known bound
O. Babaoglu et al.	Probabilistic	HeartBeat	Random Graph	None
Van Steen et al.	Probabilistic	External	Random Graph	None
Baldoni, et al.	Probabilistic	Internal	Random Graph	None

# Correct Clock (1/2)

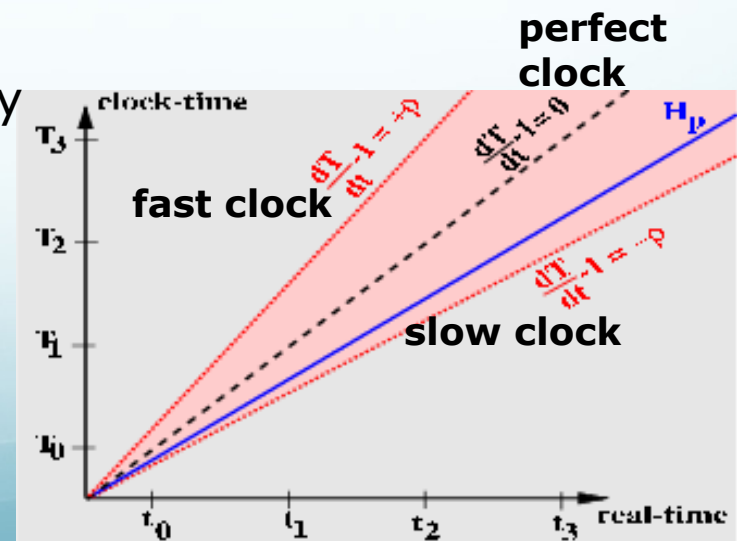
- An hardware clock  $H$  is **correct** if its drift rate is within a limited bound of  $\rho > 0$  (e.g.  $10^{-6}$  secs/ sec).

$$1 - \rho \leq dC/dt \leq 1 + \rho$$

- In presence of a correct hardware clock  $H$  we can measure a time interval  $[t, t']$  (for all  $t' > t$ ) introducing only limited errors.

$$(1 - \rho)(t' - t) \leq H(t') - H(t) \leq (1 + \rho)(t' - t)$$

- Max skew  $D$ : resynchronize at least every  $D/2\rho$  seconds.



# Correct Clock (2/2)

- Software clocks have to be monotone  
 $t' > t \rightarrow C(t') > C(t)$
- The monotonic property can be guaranteed choosing opportune  $\alpha$  and  $\beta$ .
  - Note that  $\alpha$  and  $\beta$  can be a function of time

## Clock **failure**:

- *crash failure* – clock simply stops
- *arbitrary failure* – clock behaves arbitrarily (e.g. Y2K bug: the day after the 31/1/1999 becomes 1/1/1900 rather than 1/1/2000)

Correctness is no accuracy...

# Internal Synchronization in a Synchronous System

Characteristics of a **Synchronous** system:

- Known maximum time taken for each computational step
- Known maximum clock drift
- *Known upper and lower bound for communication delay*

# Internal Synchronization in a Synchronous System

## Algorithm

1. A process  $p_1$  sends its local time  $t$  to a process  $p_2$  by mean of a message  $m$
2.  $p_2$  should set its clock to  $t+T_{\text{tran}}$  where  $T_{\text{tran}}$  is the transmission delay of message  $m$ 
  1.  $T_{\text{tran}}$  is unknown but  $\min \leq T_{\text{tran}} \leq \max$
3. We denote  $u=(\max-\min)$  the uncertainty of transmission delay
4.  $p_2$  sets its local clock to  $t+(\max+\min)/2$ 
  1. Skew is at most  $(\max+\min)/2$

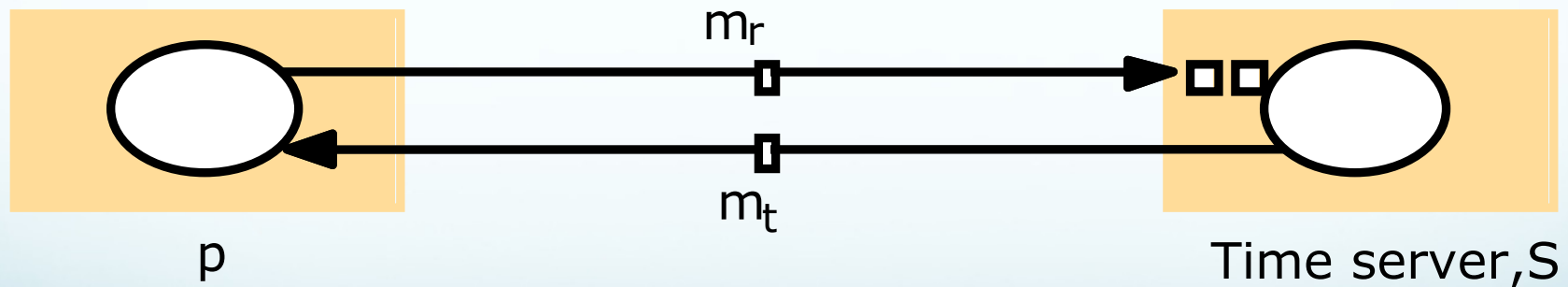
**SYNCHRONOUS SYSTEM:**  $T_{\text{tran}} = \min + x$  con  $x \geq 0$

# Synchronization by mean of a Time Server

- Centralized Time Service
  - Request-driven
    - Cristian's Algorithm
  - Broadcast-based
    - Berkeley Unix algorithm - Gusella & Zatti (1989)
- Distributed Time Service (Network Time Protocol)

# Christian's Algorithm

- A node is a time server  $S$  (presumably with access to UTC). Other nodes want to access to time server's clock.
- A process  $p$  asks the current time through a message  $m_r$  and receives  $t$  in  $m_t$  from  $S$ 
  - $p$  sets its clock to  $t + T_{round}/2$ ,  $T_{round}$  is round trip time experienced by  $p$



## Notes:

- A time server can crash
- Cristian suggests using a cluster of synchronized time servers
- A time server can be attacked...

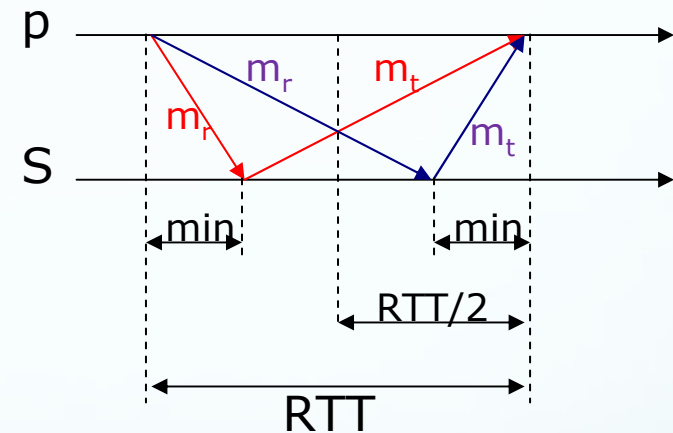
# Accuracy

## Case 1

Time employed by the response is greater than estimate obtained by  $RTT/2$ , in particular is equal to  $(RTT - min)$

$\Delta = \text{estimate of response} - \text{real time} = (RTT/2) - (RTT - min) =$

$(-RTT + 2min)/2 = -RTT/2 + min = \mathbf{-(RTT/2 - min)}$



## Case 2

Time employed by the response is greater than estimate obtained by  $RTT/2$ , in particular is equal to  $(RTT - min)$

$\Delta = \text{estimate of response} - \text{real time} = \mathbf{(RTT/2) - min} = \mathbf{+(RTT/2 - min)}$

Consequently the accuracy of Cristian algorithm is  $\pm (T_{\text{round}}/2 - min)$  where  $min$  is the minimum transmission delay

# Probability and Synchronization

- Cristian's algorithm requires the knowledge of distribution of transmission delay (mean, std. dev., etc..)
- It is possible to define a maximum error and to repeat the synchronization procedure until we are able to read the time value with the desired accuracy
  - Improve Precision
    - By taking several measurements and taking the smallest round trip
    - Or using an average after throwing out the large values
- The algorithm is probabilistic
  - *The reading of time value with the desired accuracy could not happen in a reasonable number of iterations*
  - The probability of a loss of synchronization can be computed starting from the knowledge of the distribution of transmission delay

# The Berkeley Algorithm

- Berkeley Algorithm: internal clock synchronization among small sets of computers
  - *Master* asks, through broadcast message, all other nodes for their clock values then collects the answers from other nodes (*slaves*)
  - Master utilizes experienced round trip time to estimate clock values of slaves
  - It computes a mean of obtained clock values
  - The Master tells each node how to adjust its clocks (if the adjustment requires local clocks go back in time, slaves simply slow down their clocks)

# The Berkeley Algorithm

- A desired accuracy is obtained throwing out clock values obtained with RTTs greater than a predefined maximum value
- Fault tolerance:
  - When a master crashes another node is elected master (in unbounded time)
  - The Berkeley Algorithm is tolerant with respect to arbitrary behavior. Master computes means using only a subset of collected clock values with respect to the whole set of slaves.

# Berkeley Algorithm: esempio



$$T_{S1} = 690$$

$$T_{\text{new}} = 690 + 5 = 695$$

$$T_{\text{new}} = 700 - 5 =$$



$$T_M = 700$$



$$T_{S2} = 705$$

$$T_{\text{new}} = 705 - 10 = 695$$



$$T_{S3} = 695$$

$$T_{\text{new}} = 695 + 0 = 695$$

$$d_1 = 690 - 700 = -10$$

$$d_2 = 705 - 700 = 5$$

$$d_3 = 695 - 700 = -5$$

$$\text{Avg} = (0 - 10 - 5 + 5) / 2 = -5$$

$$\text{Adj}_{S1} = -5 + 10 = 5$$

$$\text{Adj}_{S2} = -5 - 5 = -10$$

$$\text{Adj}_{S3} = -5 + 5 = 0$$

$$\text{Adj}_M = -5 + 0 = -5$$

# Berkeley Algorithm

## **Osservazione:**

What does slowing down a clock mean?

- We cannot impose a clock value in past in slaves that have a clock value greater than the new computed mean.
  - This action can violate the cause/effect ordering of the events produced by the slave and the time monotonicity.
- Consequently we slow down clocks hiding interrupts.
  - Hiding interrupts, the local clock is not updated so that we have to hide a number of interrupt equals to slowdown time divides the interrupt period.

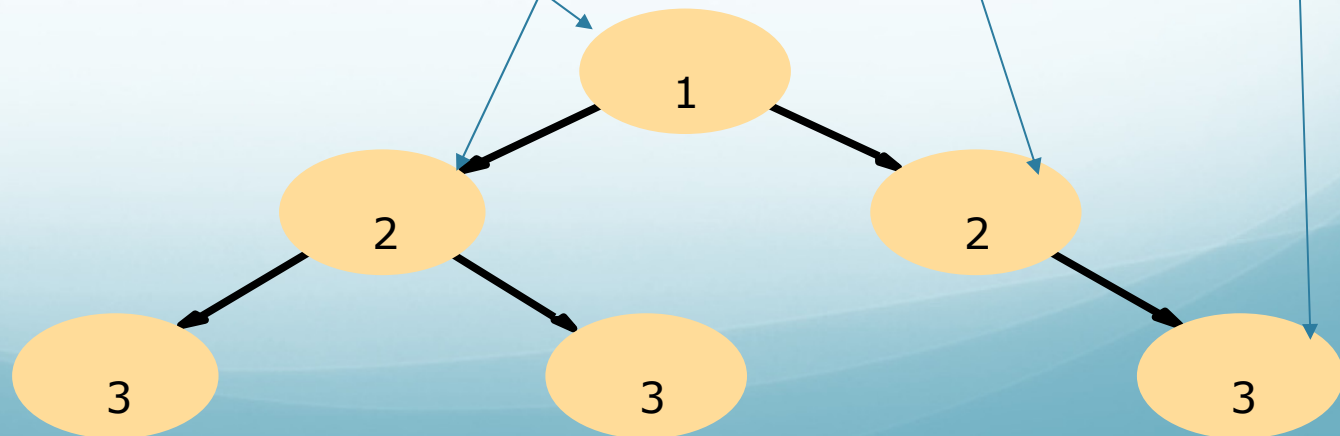
# Network Time Protocol (NTP)

- Time service over **Internet** - synchronizes clients with UTC:
- Reliability by mean of redundant server and path
- Scalable

Primary servers are connected to UTC sources

Secondary servers are synchronized to primary servers

Synchronization subnet - lowest level servers in users' computers



# Network Time Protocol

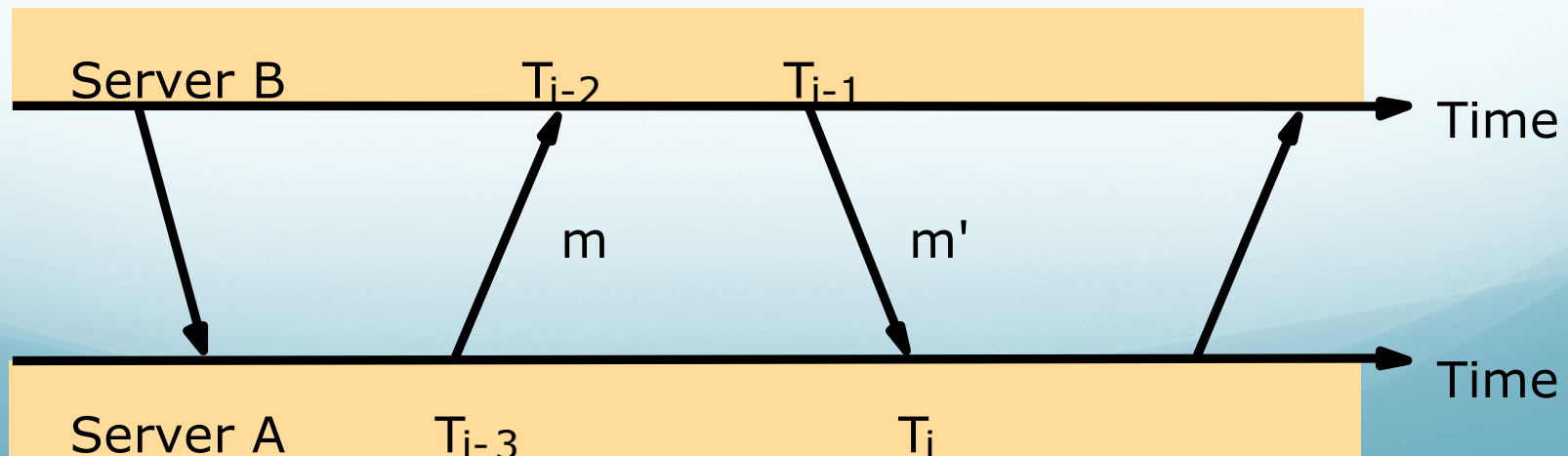
- Synchronization of clients relative to UTC on an Internet-wide scale
  - NTP is a *standard de facto* for **external** clock synchronization of distributed system on Internet
  - NTP employs several security mechanisms (e.g. mechanisms for authentication of time references) usually they are not required in a local area network
- Based on a remote reading procedure like Cristian's Algorithm
  - NTP specification adds to the basic algorithm mechanisms for clustering, filtering and evaluating data quality in order to minimize the synchronization

# NTP – Server Synchronization

- **The NTP hierarchy is reconfigurable in presence of faults**
  - Primary server that loses its connection with UTC-signal can become a secondary server
  - Secondary server that loses its connection with a primary server (e.g. a crash of the primary server) can contact and connect itself to another primary.
- **Synchronization Modes:**
  - **Multicast:** server periodically sends its actual time to its leaves in the LAN. Leaves set their time using the received time assuming a certain delay. It is used in quick LANs but it shows a low accuracy
  - **Procedure call:** server replies to requests with its actual timestamp (like Cristian's algorithm). High Accuracy and it is useful when it is not available hw multicast.
  - **Symmetrical:** used to synchronize between pairs of time servers using messages containing timing information. Only used in high level of hierarchy.

# Message exchange

- In all cases the UDP is used
- Each message contains timestamps of recent events:
  - Local times of the *Send* and the *Receive* of previous message
  - Local time of the *Send* of current message
- Receiver stores the arrival time of the message  $T_i$  that contains the timestamps  $T_{i-3}$ ,  $T_{i-2}$ ,  $T_{i-1}$



# Accuracy of NTP

- For each pair of message exchanged between two nodes(server and client) the NTP daemon estimates the offset  $o_i$  between the 2 local clocks and a delay  $d_i$  (total transmission time for the request-reply message exchange)
- Assuming that the real offset between clock values of B and A is  $o$  and that transmission delays of messages  $m$  and  $m'$  are respectively  $t$  e  $t'$

$$T_{i-2} = T_{i-3} + t + o \text{ and } T_i = T_{i-1} + t' - o$$

- Consequently the total transmission delay is:

$$d_i = t + t' = T_{i-2} - T_{i-3} + T_i - T_{i-1}$$

# Accuracy of NTP

- Consequently we have

$$\mathbf{o} = \mathbf{o}_i + (\mathbf{t}' - \mathbf{t})/2 \text{ where } \mathbf{o}_i = (\mathbf{T}_{i-2} - \mathbf{T}_{i-3} + \mathbf{T}_i - \mathbf{T}_{i-1})/2$$

- Moreover considering  $t, t' > 0$  it is possible to prove that

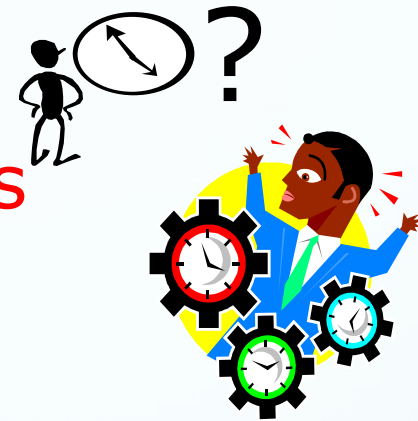
$$\mathbf{o}_i - \mathbf{d}_i / 2 \leq \mathbf{o} \leq \mathbf{o}_i + \mathbf{d}_i / 2 .$$

- $O_i$  is an offset estimate and  $d_i$  is a measure of accuracy of the estimate
- NTP servers filters pairs  $\langle o_i, d_i \rangle$ , estimating the accuracy of data and using  $o_i$  with smallest  $d_i$  (the smaller delay the better accuracy)
- Accuracy: 10 millisecs on Internet paths (1 within LANs)
- **Important Observation:**
  - **Note that the reading error does not from the absolute value of RTT but from the difference between the transmission times  $t$  and  $t'$ . RTT is only an estimate of the error! If we have symmetrical communication paths for each RTT value the reading error is zero!**

# Time in Asynchronous Systems

Physical Time: A global property...  
Observable?

**NO** in a distributed asynchronous system: different clocks are synchronizable only with a certain probability



- The impossibility of perfect accuracy is due to unpredictability of communication delay.
  - We can introduce a bound for the accuracy only when we know the upper and lower bounds for communication delays.