Clock Synchronization in a Distributed System

Topics

- **Introduction**
- **Clock synchronization**
- **Physical Clock**
- **Logical clocks**
- **Election algorithms**

Synchronized Distributed Clocks

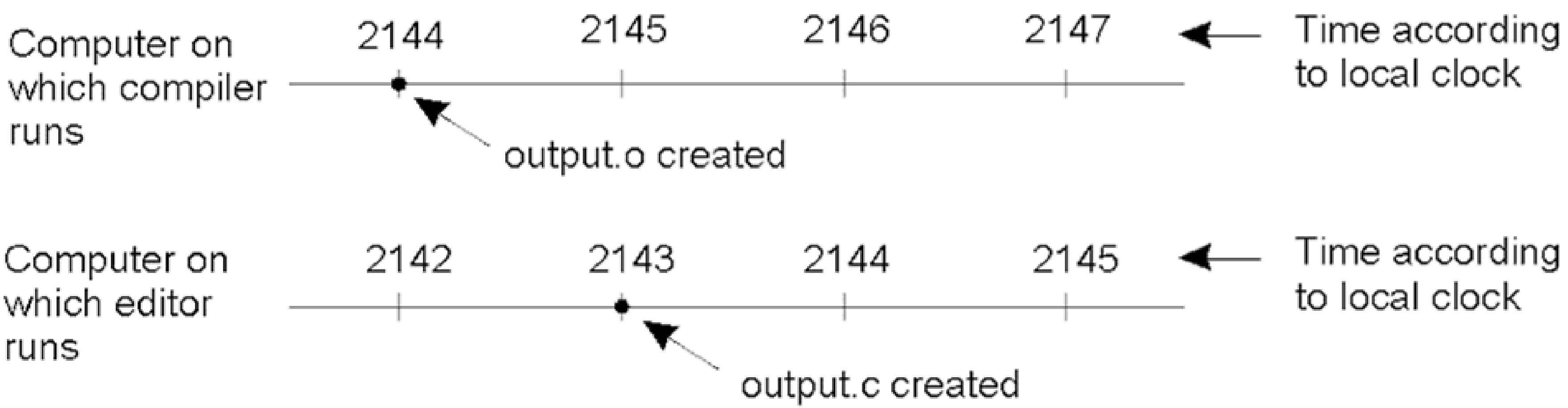
NEED:

- ***Time driven systems:*** in statically scheduled systems activities are started at "precise" times in different points of the distributed system.
- ***Time stamps:*** certain events or messages are associated with a time stamp showing the actual time when they have been produced; certain decisions in the system are based on the "exact" time of the event or event ordering.
- ***Calculating the duration of activities***: if such an activity starts on one processor and finishes on another (e.g. transmitting a message), calculating the duration needs clocks to be synchronized.

Lack of Global Time in DS

- It is impossible to guarantee that physical clocks run at the same frequency
- Lack of global time, can cause problems
- Example: UNIX make
  - Edit output.c at a client
  - output.o is at a server (compile at server)
  - Client machine clock can be lagging behind the server machine clock

Lack of Global Time – Example



When each machine has its own clock, an event that occurred after another event may nevertheless be assigned an earlier time.

Physical Clock

- Every computer is equipped with CMOS clock circuit. These are electronic devices that count oscillations occurring in a crystal.
- Also called timer, usually a quartz crystal, oscillating at a well defined frequency.
- Timer is associated with two registers:
  - **A Counter and**
  - **a Holding Register**,
- Counter decreasing one at each oscillations.When counter reaches zero, an interrupt is generated; this is the ***clock tick***. And again counter is loaded with the value of holding register
- Clock tick have a frequency of 60-100 ticks per second.

Drifting of Clock (cont'd)

- The problems:

1. Crystals cannot be tuned perfectly. Temperature and other external factors can also influence their frequency.

↓

<u>Clock drift</u>: the computer clock differs from the real time.

2. Two crystals are never identical.

↓

<u>Clock skew</u>: the computer clocks on different processors of the distributed system show different time.

Need for Precision Time

- Social networking services
- Stock market buy and sell orders
- Secure document timestamps (with cryptographic certification)
- Aviation traffic control and position reporting
- Radio and TV programming launch and monitoring
- Intruder detection, location and reporting
- Multimedia synchronization for real-time teleconferencing
- Interactive simulation event synchronization and ordering
- Network monitoring, measurement and control
- Early detection of failing network infrastructure devices and air conditioning equipment
- Differentiated services traffic engineering
- Distributed network gaming and training

Clock synchronization

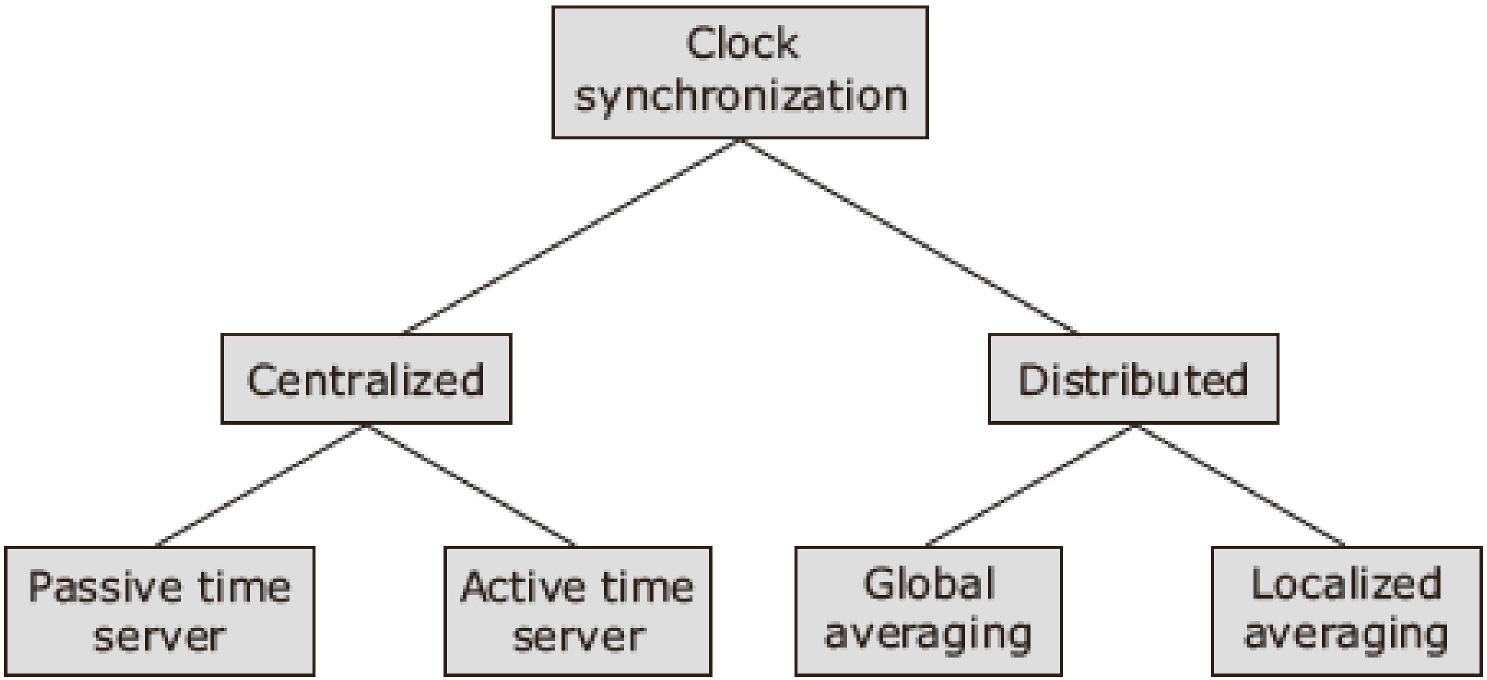


Figure 5-6 Classification of clock synchronization algorithms

Clock Synchronization Algorithms

☞ ***Distributed Algorithms***

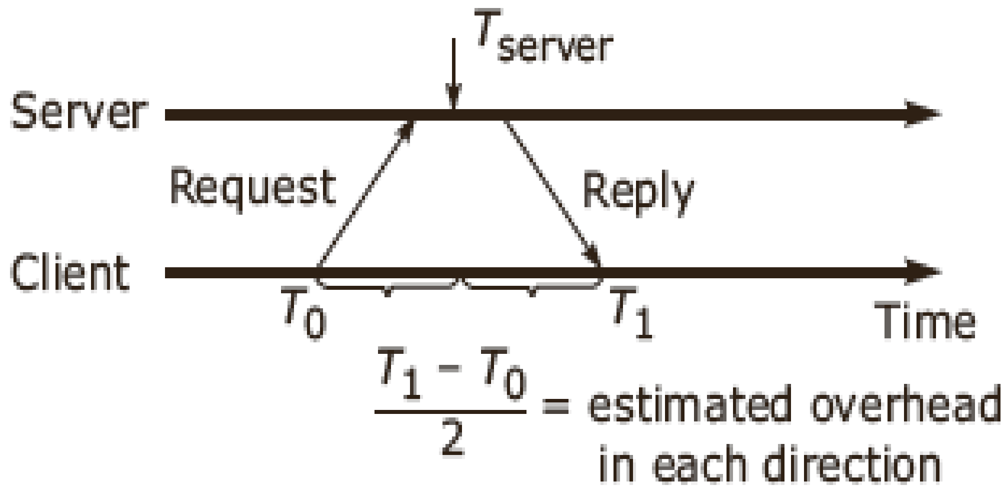
- There is no particular time server.
- The processors periodically reach an agreement on the clock value by averaging the time of neighbors clock and its local clock.
  - This can be used if no UTC receiver exists (no external synchronization is needed). Only internal synchronization is performed.
  - Processes can run on different machines and no global clock to judge which event happens first.

Cristian's Algorithm

- *Cristian's Algorithm is centralized algorithm.*

- The simplest algorithm for setting time, it issues a Remote Procedure Call to time server and obtain the time.
- A machine sends a request to time server .
- The time server sends a reply with current UTC when receives the request.
- The machine measures the time delay between time server sending the message and machine receiving it. Then it uses the measure to adjust the clock.

Centralized algorithm



Client sets time to: $T_{new} = T_{server} + \frac{T_1 - T_0}{2}$

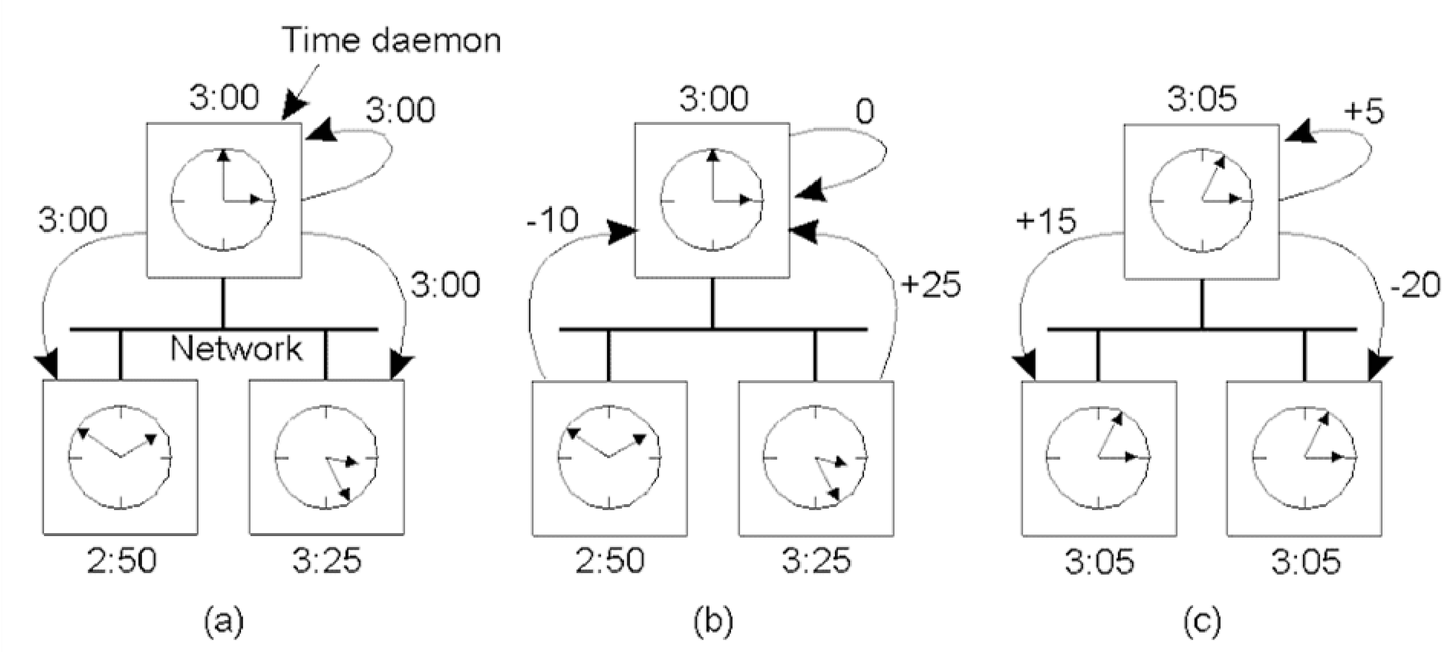
Figure 5-8 Time approximation using passive time server algorithm

Berkeley Algorithm

- ***It is also a Centralized algorithm and its Time server is an Active Machine.***

- The server polls each machine periodically, asking it for the time.
- When all the results are in, the master computes the average time.
- Instead of sending the updated time back to slaves, which would introduce further uncertainty due to network delays, it sends each machine the offset by which its clock needs adjustment.
- If master machine fails, any other slave could be elected to take over.

Berkeley Algorithm



a) The time daemon sends synchronization query to other machines in group.
b) The machines sends timestamps as a response to query.
c) The Server averages the three timestamps and tells everyone how to adjust their clock by sending offsets.

Berkeley's Algorithm

Algorithm

Elect* the master amongst N nodes. Let T_m be the time estimate of the master's clock.

Let $t[i]$ contain the time at each i slave at master

If master

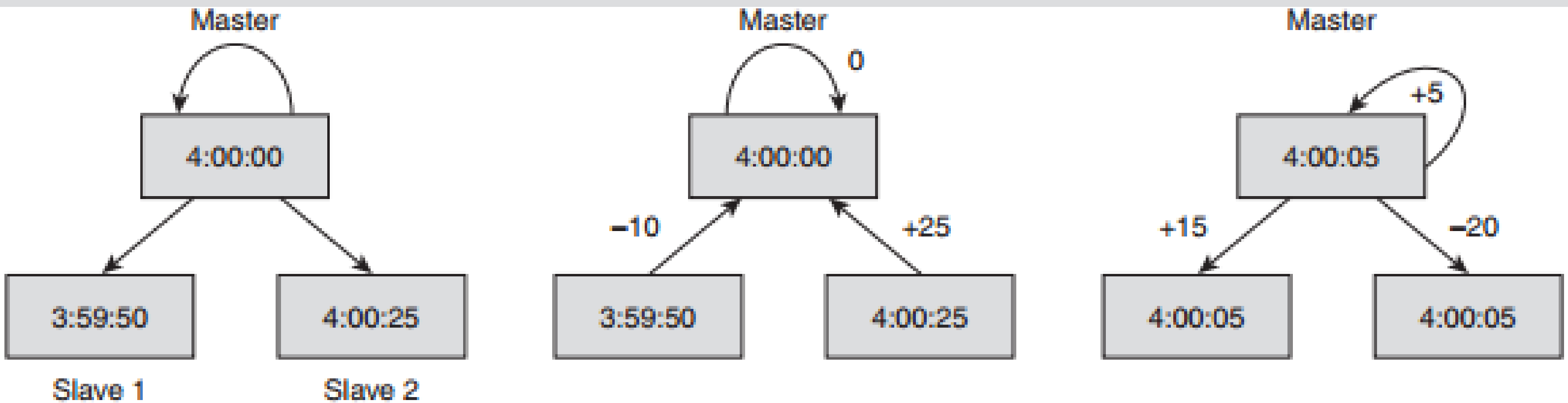
send its T_m along with query for $t[i]$ to slaves; /* for $i = 1.....N-1$ */

Adjust = Sum($t[i]$)/N /* take average including masters

send offset[i] = Adjust-$t[i]$ to each slave;

If slave

sends query response as $t[i] = T_m - T[i]$; /* for $i = 1..N-1$; calculates the difference between master timestamp T_m, and its own timestamp T */



(a) Query-poll (b) Response (c) Adjust

Disadvantages of centralized clock

- Subjected to single point failure
- From scalability point of view it is generally not acceptable to get the time requests serviced by single time server.

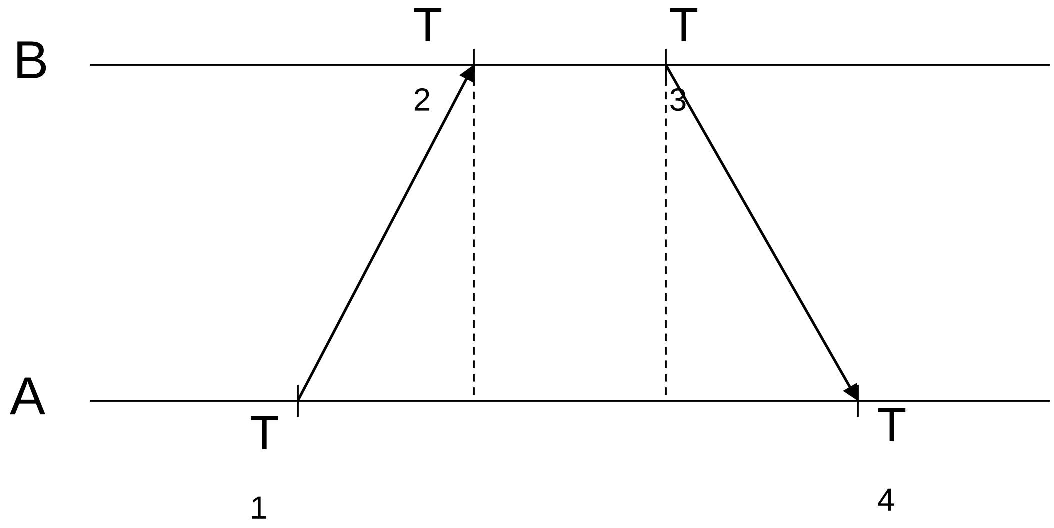
Network Time Protocol

- NTP is an application layer protocol
- Default port number 123
- Uses standard UDP Internet transport protocol
- Adjust system clock as close to UTC as possible over the Internet.
- Enable sufficiently frequently resynchronizations. (scaling well on large num
- The NTP service is provided by a network of servers located across the Internet.

Network Time Protocol

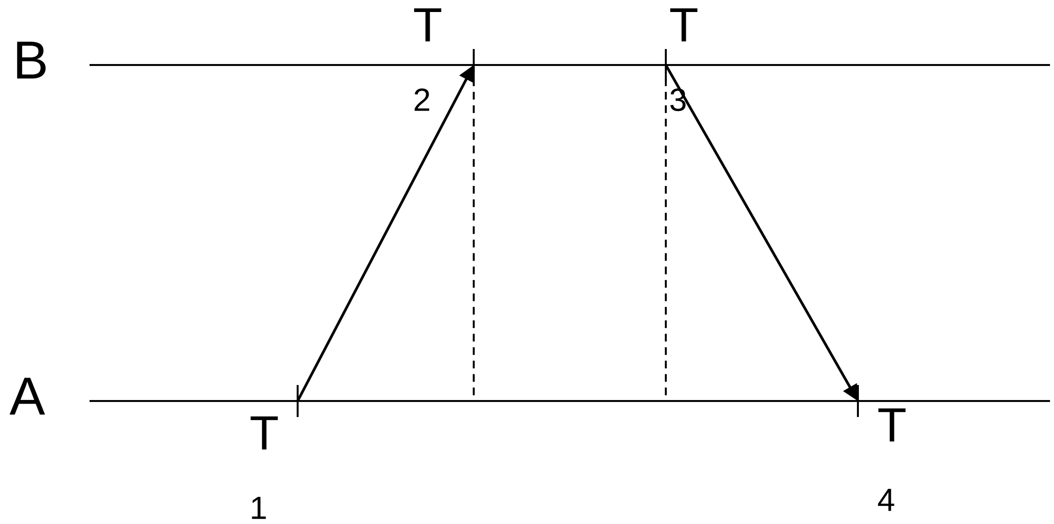
- Primary servers are connected directly to a time source. (e.g. a radio clock receiving UTC, GPS). bers of clients and servers)
- Secondary servers are synchronized with primary servers.
- The servers are connected in a logical hierarchy called a synchronization subnet. Each level of the synchronization subnet is called stratum.

NTP (Network Time Protocol)-2



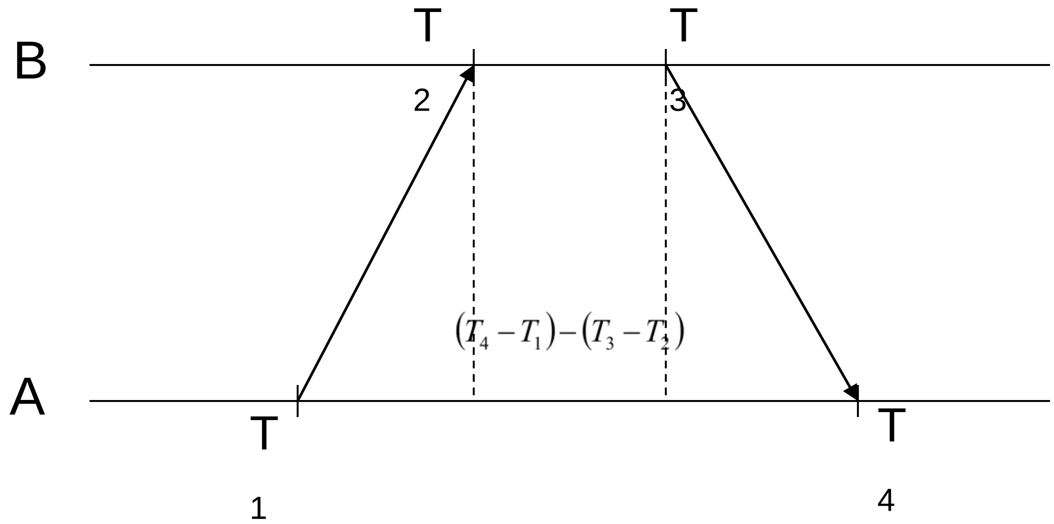
- *A* requests time of *B* at its own T_1
- *B* receives request at its T_2, records
- *B* responds at its T_3, sending values of T_2 and T_3
- *A* receives response at its T_4
- Question: what is $\theta = T_B - T_A$?

NTP (Network Time Protocol)-3



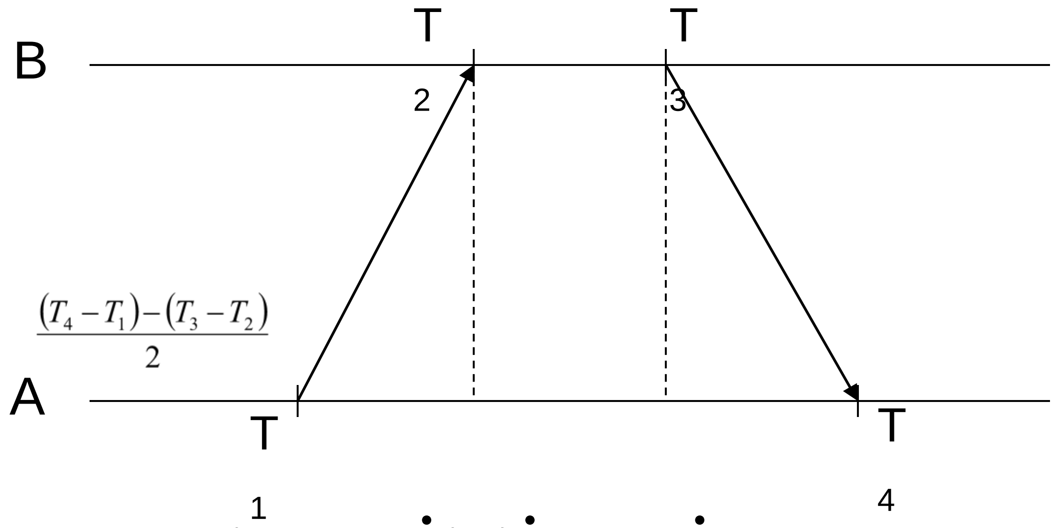
- Question: what is $\theta = T_B - T_A$?
- Assume transit time is approximately the same both ways
- Assume that *B* is the time server that *A* wants to synchronize to

NTP (Network Time Protocol)-4



- A knows $(T_4 - T_1)$ from its own clock
- B reports T_3 and T_2 in response to NTP request
- A computes total transit time of

NTP (Network Time Protocol)-5



- One-way transit time is approximately ½ total, i.e.,
- *B*'s clock at T_4 reads approximately

$$T_3 + \frac{(T_4 - T_1) - (T_3 - T_2)}{2}$$

NTP -6

- Servers organized as *strata*
  - *Stratum 0* server adjusts itself to WWV directly
  - *Stratum 1* adjusts self to *Stratum 0* servers
  - Etc.
- Within a stratum, servers adjust with each other

Summary

- Real synchronization is imperfect.
- Clocks never exactly synchronized.
- Often inadequate for distributed systems
  - might need totally-ordered events
  - might need millionth-of-a-second precision

Logical Clocks

Logical Clock

- Assume no central time source –
  - Each system maintains its own local clock .
  - Allow to get global ordering on events.

- Assign sequence numbers to messages –
  - *All cooperating processes can agree on order of event.*

Lamport's Timestamps

- It is used to provide a partial ordering of events with minimal overhead.
- It is used to synchronize the logical clock.
- **It follows some simple rules:**
  - ❖A process increments its counter before each event in that process i.e. Clock must tick once between every two events.
  - ❖When a process sends a message, it includes its timestamp with the message.
  - ❖On receiving a message, the receiver process sets its counter to be the maximum of the message counter and increments its own counter .

'Happened Before' Relation

- a *'Happened Before'* b : **a→b**

Situations:

1. If a and b are events in the same process, and a comes before b, then $a \rightarrow b$.

1. If

   a :event of message sent

   b : event of receipt of the same message

   then $a \rightarrow b$.

2. **HBR is Transitive**:

   If $a \rightarrow b$ and $b \rightarrow c$ then $a \rightarrow c$.

Note: Two distinct events a and b happens in different process that do not exchange messages then these events are said to be *concurrent*

a -/->b and b -/->a

Implementation of Logical Clocks

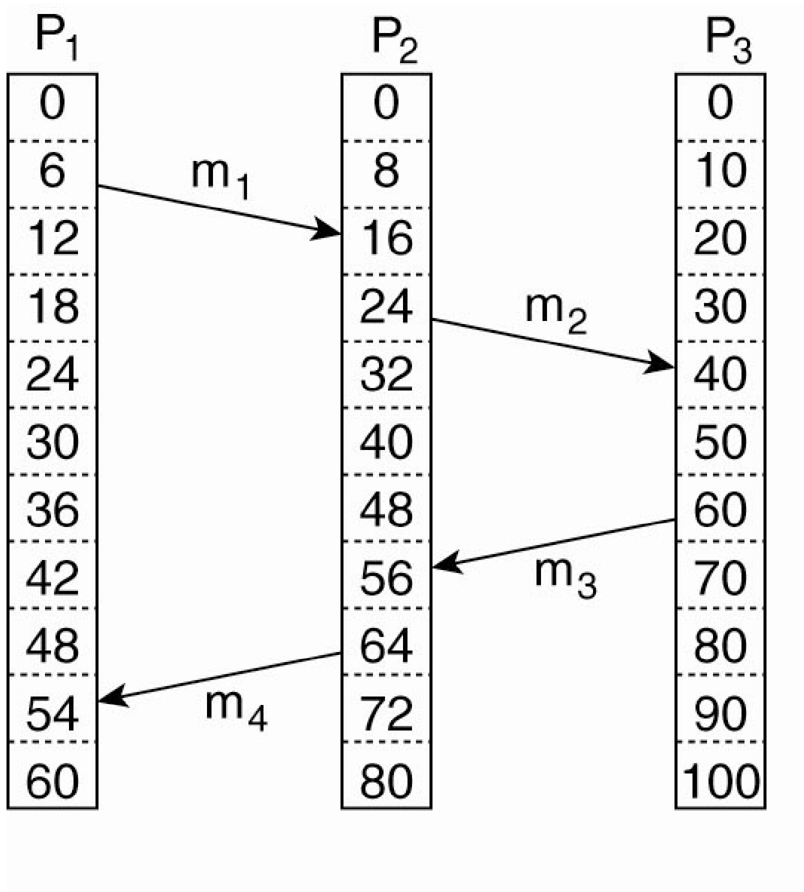
Conditions for correct functioning:

- C1: If a and b are two events in the same process, and a→ b, then we demand that C(a) < C(b).
- C2: If a corresponds to sending a message m, and b corresponds to receiving that message, then also C(a) < C(b).
- C3: A clock C associated with the process P must always go forward, never backwards. Hence corrections to a logical clock must be always made by adding a positive value , never subtracting from it.

Lamport logical clocks

- Lamport clock *L* orders events consistent with logical "happens before" ordering
  - If e $\rightarrow$ e', then $L(e) < L(e')$
- But not the converse
  - $L(e) < L(e')$ does not imply $e \rightarrow e'$
- Similar rules for concurrency
  - $L(e) = L(e')$ implies $e \parallel e'$ (for distinct e,e')
  - $e \parallel e'$ does not imply $L(e) = L(e')$
- i.e., Lamport clocks arbitrarily order some concurrent events

Lamport's Logical Clocks (2)



(a)

Three processes, each with its own clock. The clocks run at different rates.

Lamport's algorithm

- Each process *i* keeps a local clock, L_i
- Three rules:

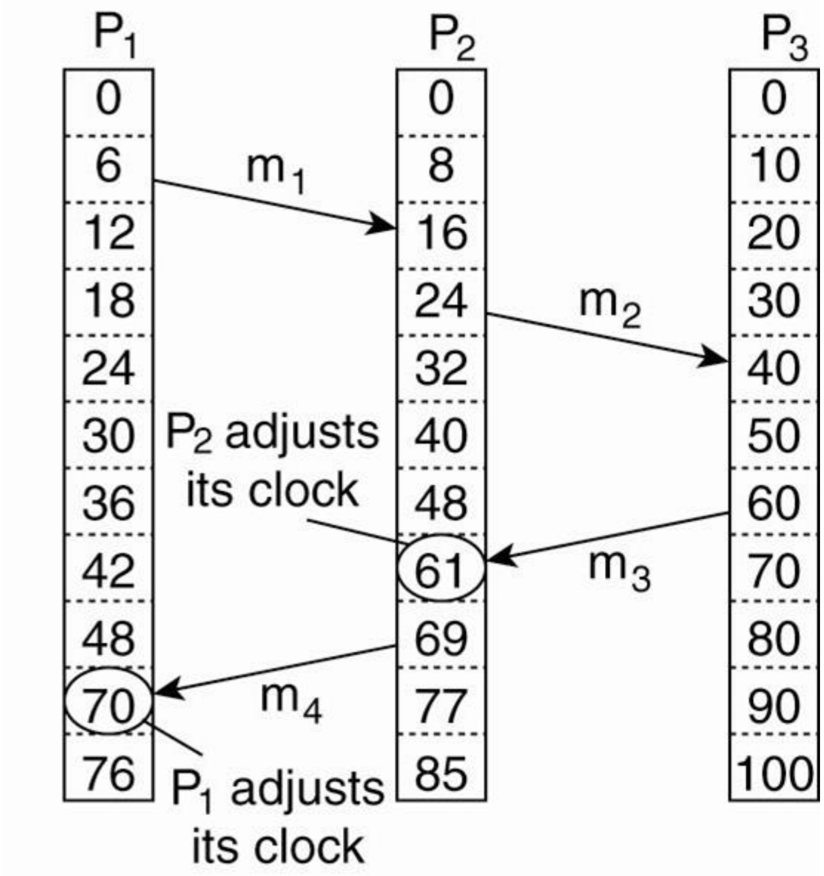
1. At process *i*, increment L_i before each event
2. To send a message *m* at process *i*, apply rule 1 and then include the current local time in the message: i.e., $send(m,L_i)$
3. To receive a message (m,t) at process *j*, set $L_j = max(L_j,t)$ and then apply rule 1 before time-stamping the receive event

- The global time $L(e)$ of an event *e* is its local time
  - For an event *e* at process *i*, $L(e) = L_i(e)$

Rules for adjusting clocks

- For two events a and b in same process p1
  - $C(b)= C(a)+1$
- If a is sending process and b is receiving process of pi and pj then,
  - $Cj(b)=max((Ci(a)+1),Cj(b))$

Lamport's Logical Clocks



(b)

Lamport's algorithm corrects the clocks.

Position of logical clocks in Middleware

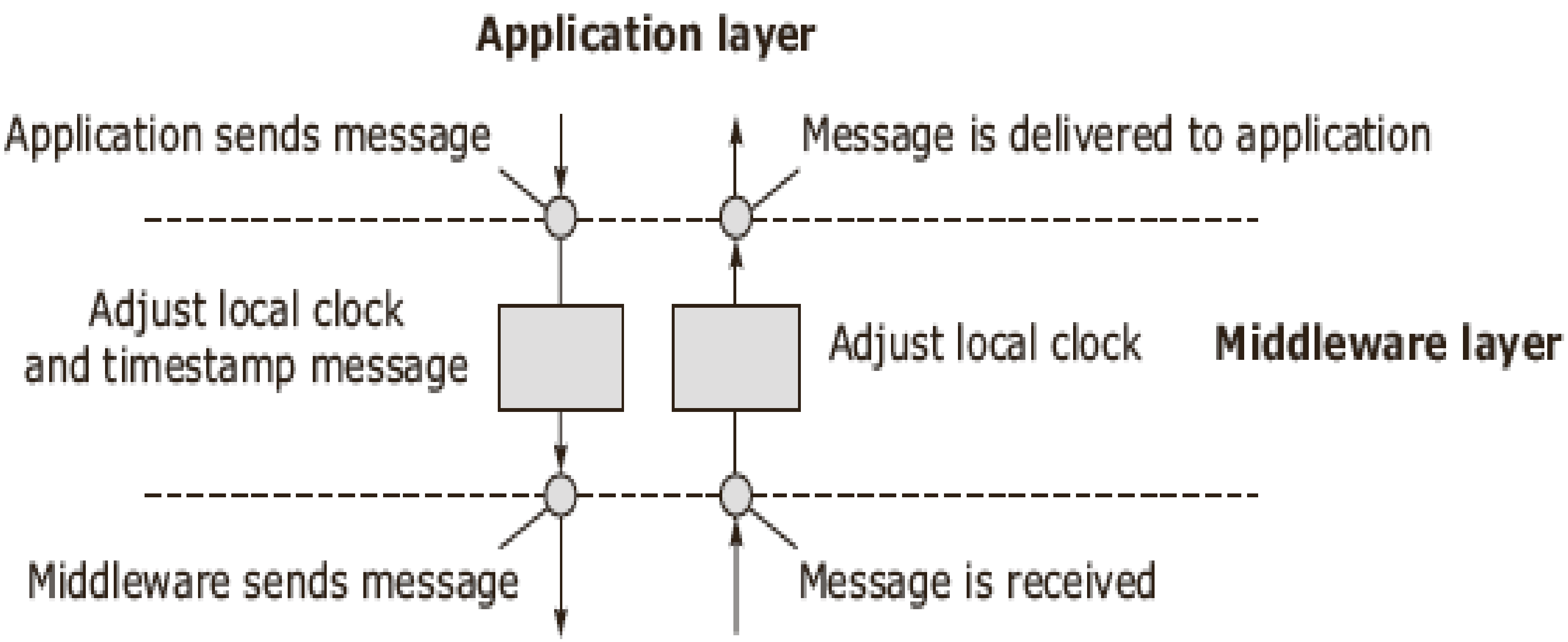


Figure 5-18 Positioning of Lamport's logical clocks in distributed systems

Vector Timestamps – 1

- In the system of vector clocks, the time domain is represented by a set *non-negative integer vectors.*
- Each process pi maintains a vector vti[1..n], where vti[i] is the local logical clock of pi and describes the logical time progress at process pi.
- Vti[j]represents process pi’s latest knowledge of process pj local time. If vti[j] = x, then process pi knows that local time at process pj has progressed till x.
- The entire vector vti constitutes pi’s view of the global logical time and is used to timestamp events.

Vector Timestamps – 2

- Initially, all vectors [0,0,…,0]
- For event on process i, increment own c_i
- Label message sent with local vector
- When process j receives message with vector $[d_1, d_2, \ldots, d_n]$:
  - Set local each local entry k to $\max(c_k, d_k)$
  - Increment value of c_j

Vector Timestamps – 3

- Vector clocks overcome the shortcoming of Lamport logical clocks
  - *L*(e) < *L*(*e*') does not imply *e* happened before *e*'
- Vector timestamps are used to timestamp local events
- They are applied in schemes for replication of data

Vector Timestamps – 4

Process pi uses the following two rules **R1 and R2 to update its clock:**

• **R1: Before executing an event, process pi updates its local logical time as follows:**

$$vt_i[i] := vt_i[i] + d \qquad (d > 0).$$

• **R2 Each message *m is piggybacked with the vector clock vt of the sender* process at sending time**.

On the receipt of such a message *(m,vt), process* pi executes the following sequence of actions:

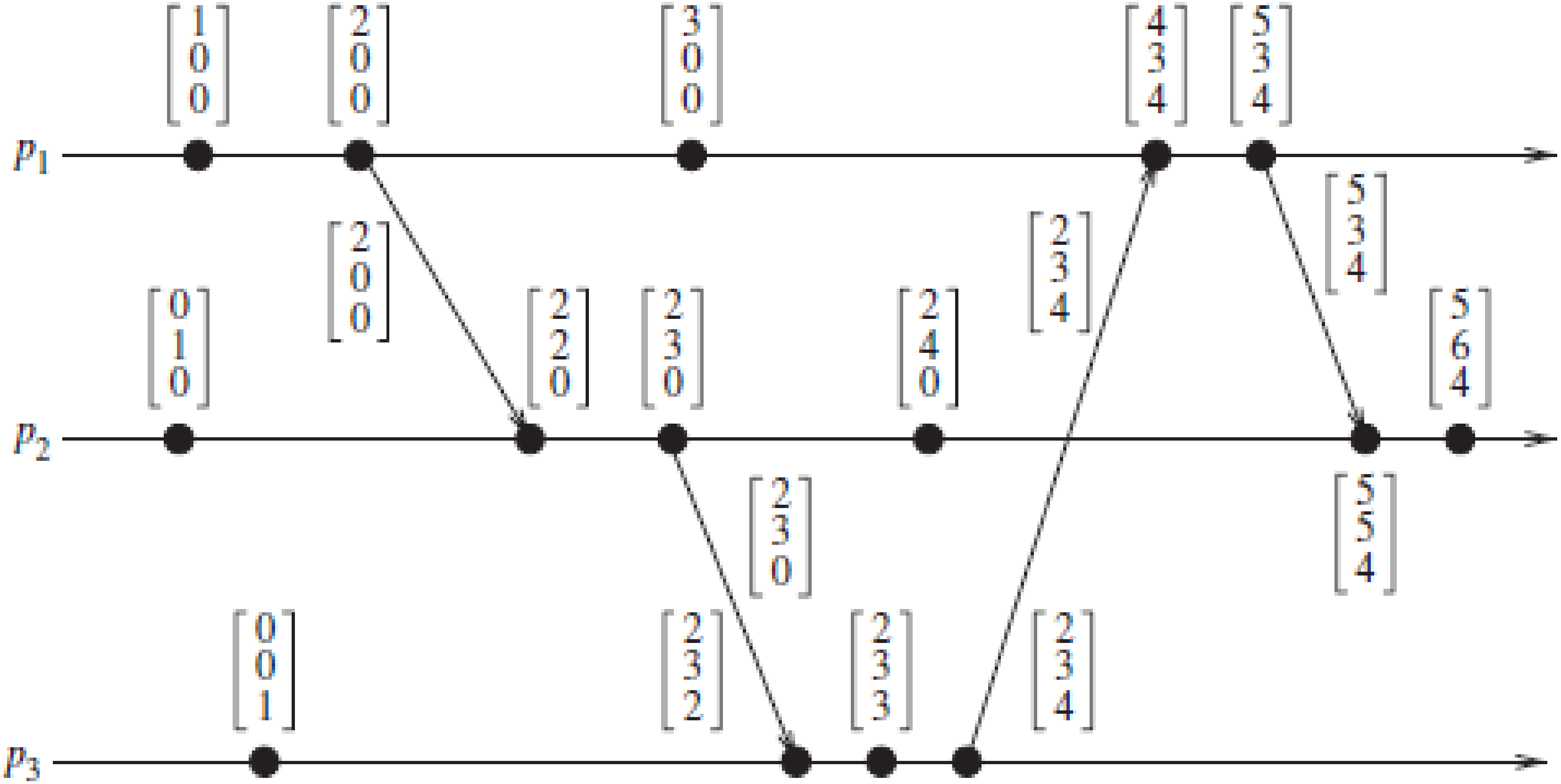
1. update its global logical time as follows:

$$1 \leq k \leq n \ : \ vt_i[k] := max(vt_i[k], vt[k]);$$

2.execute **R1;**

3. deliver the message m.

Vector Timestamps – 5(Example)



Vector Timestamps – 6

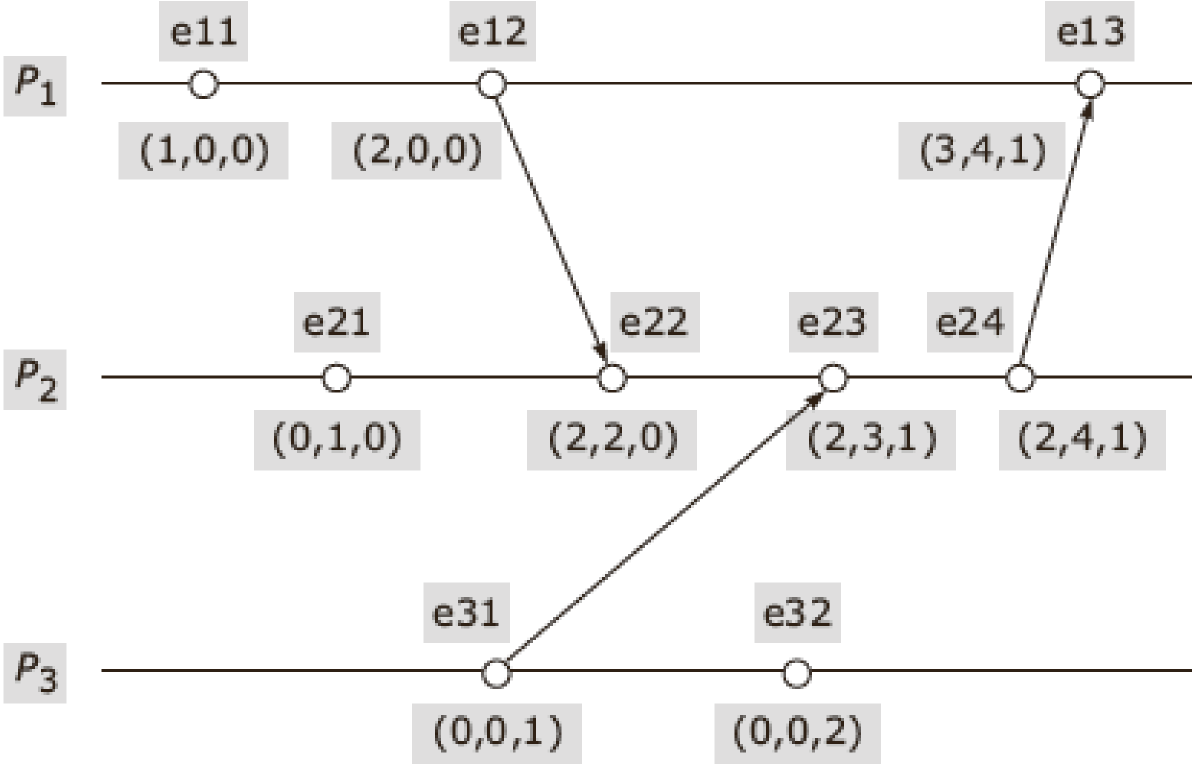


Figure 5-20 Example of vector timestamp

Important Lessons

- Clocks on different systems will always behave differently
  - Skew and drift between clocks
- Time disagreement between machines can result in undesirable behavior
- Two paths to solution: synchronize clocks or ensure consistent clocks
- Clock synchronization
  - Rely on a time-stamped network messages
  - Estimate delay for message transmission
  - Can synchronize to UTC or to local source