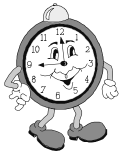
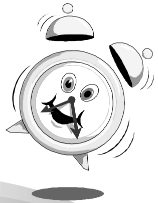


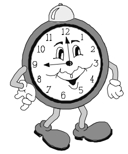
Time Service



- ➢ Requirements & problems
- ➢ Clock Compensation
- ➢ Physical Clock Synchronisation Algorithms

Requirements

- Measure delays between distributed components
- Synchronise streams e.g. sound and vision
- Detect event ordering for causal analysis
- Utilities use modification timestamps e.g. archive, make

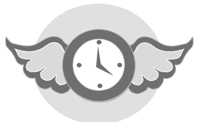


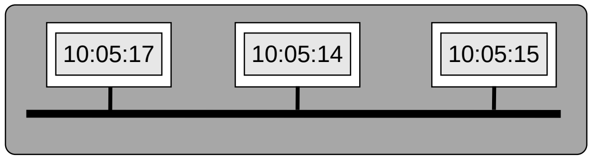
Local Time Service

- ➢ Quartz crystal oscillates and decrements counter
- ➢ On zero, counter is reset to the value in clock register and causes an interrupt.
- ➢ Interrupt rate controlled by value in register.
- ➢ Interrupt handler updates software clock e.g. secs since 1/1/1970
- ➢ Provide calls to read, compare, convert to and from printable time sec:min:hours:day:month:year

Problems

- A clock's frequency varies with temperature
- Clocks on different computers drift due to differing oscillation period





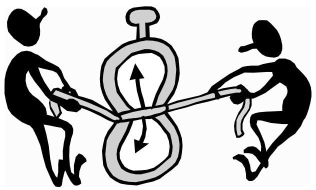
- ➢ Typical accuracy is 1 in 10^{-6} = 1 sec in 11.6 days

- ➢ Centralised time service?
- ➢ Impractical due to variable message delays

Time Sources

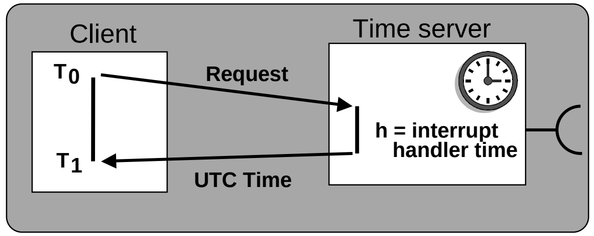
- ➢ Universal Coordinated Time (UTC)
  Based on atomic clocks but leap seconds inserted to keep in phase with astronomical time - earth's orbit round sun.
- ➢ Radio stations broadcast UTC & provide a short pulse every second. Random atmospheric delays make accuracy ±10 msec
- ➢ Geostationary Environment Operation Satellite (GEOS) or Global Positioning Systems (GPS) provide UTC to ±0.5msec
- ➢ Require (GPS or UTC) receivers on servers to support a clock synchronisation service.

Clock Compensation



- Assume 2 clocks can each drift at rate of r msec/s.
  Max difference = 2r msec/s
  To guarantee accuracy between 2 clocks to within d msecs requires resynch every d/2r secs.
- Get UTC and correct software clocks
  What happens if local clock is 5 secs fast and you set it right?
  Time must never run backward!
  Rather slow clock down so that it is reset over a period.
- Clock register normally set to generate interrupts every 10msec and interrupt handler adds 10msec to software clock.
  Instead add 9 until correction is made or add 11 to advance clock.

Cristian's Algorithm

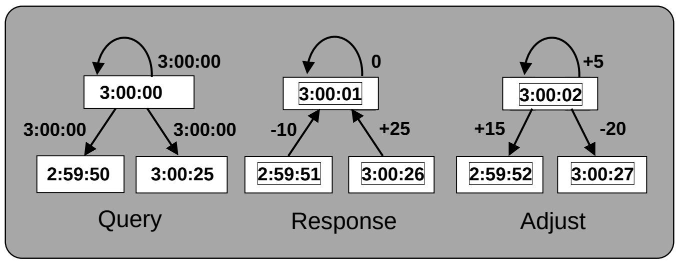


Time Server with UTC receiver gives accurate current time

- Estimate of message propagation time p = (T**1** - T**0** - h)/2
- Set clock to UTC + p
- Measure T**1** - T**0** over a number of transactions but discard any that are over a threshold as being subject to excessive delay or take minimum values as being most accurate
- Single point of failure & bottleneck
- Could broadcast to a group of synchronised servers
- An impostor or faulty server sending incorrect times can wreak havoc
  need authentication

Berkley Algorithm

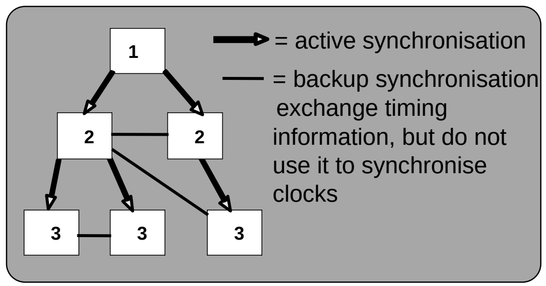
- Co-ordinator chosen as master & periodically polls slaves to query clocks.
- Master estimates local times with compensation for propagation delay
- Calculate average time, but ignore occasional readings with propagation delay greater than a cut-off value or whose current clock is badly out of synch.
- Sends message to each slave indicating clock adjustment



Synchronisation feasible to within 20-25 msec for 15 computers, with drift rate of 2×10^{-5} and max round trip propagation time of 10 msec.

Network Time Protocol (NTP)

- Multiple servers across the Internet
- Primary servers are directly connected to UTC receivers
- Secondary Servers synchronise with primaries
- Tertiary Servers synchronise with secondary servers etc.
  – less accurate due additional errors at each level.
- Scales to large numbers of servers and clients



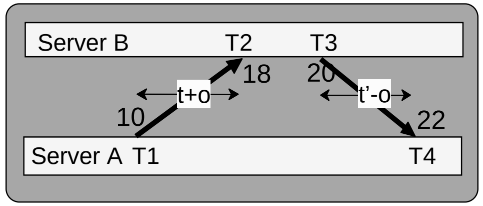
Copes with failures of servers – e.g. if primary's UTC source fails it becomes a secondary, or if a secondary cannot reach a primary it finds another one.

Authentication used to check that time comes from trusted sources

NTP Synchronisation Modes

- **Multicast**
  1 or more servers periodically multicast to other servers on high speed LAN.
  They set clocks assuming some small delay.
- **Procedure Call Mode**
  Similar to Cristian's algorithm. A client requests time from a few other servers.
  Used where there is no multicast or higher accuracy is needed e.g. a group of file servers on a LAN
- **Symmetric protocol**
  Used by master servers on LANs, and layers closest to primaries highest accuracy based on pairwise synchronisation.

NTP Symmetric Protocol



t = transmission delay (e.g. 5 ms)
o = clock offset of B relative to A (e.g. 3 ms)
Assume T1 = 10 then T2 = 18, and T3 = 20 then T4 = 22
Let a = T2 - T1 = **t** + **o,** Let b = T4 - T3 = **t'** - **o**
Round trip delay = **t** + **t'** = a + b = (T2 - T1) + (T4 - T3)
= 18-10 + 22-20 = 10
2**o** = a - b = (T2-T1) - (T4 - T3) + (**t - t'**) = (T2-T1) - (T4 - T3) = 8-2 = 6
Clock offset **o** = (a-b)/2 = ((T2-T1) - (T4 - T3))/2 = 3 (assuming t ≈ t)

NTP Symmetric Protocol

- T4 = current message receive time is determined at receiver
- Every message contains:
  T3 = current message send time
  T2 = previous received message receive time
  T1 = previous received message send time
- Data filtering: values of **o** which correspond to minimum values of **t** are used to get average values of actual clock offset.
- Peer selection: exchange messages with several peers looking for most reliable values favouring lower level ones (e.g. primaries)
- 20-30 primaries and over 2000 secondaries can synchronise to within 30ms.

Logical Time

- For many purposes it is sufficient that processes *agree* on the same time (i.e. internal consistency) which need not be real or UTC time.

Event Ordering

- a → b = a happens before b
  1. If a and b are events in the same process and a occurs before b then a → b is true
  2. If is a is the event of message sent from process A and b is the event of message receipt by process B then a → b is true
  3. If a → b and b→ c then a → c
  4. If x and y happen in different processes which do not exchange messages then x → y is not true and y → x is not true ie x and y are said to be **concurrent** and nothing can be said about their order.
- Logical time denotes causal relationship but the → relationship may not reflect real causality
  E.g. a process may receive message x and then send message y so x → y even though it would have sent y if x had not been received.

Logical Clocks

- A monotonic software counter can be used to implement logical clocks. Each process p keeps its own logical clock Cp which it uses to timestamp events

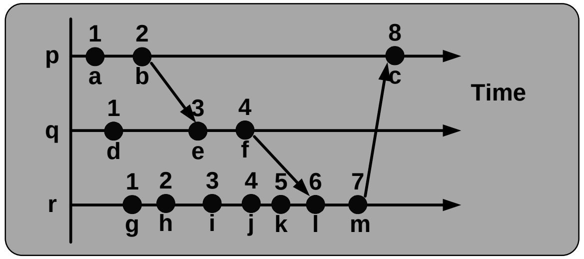
  1. Cp is incremented before assigning a timestamp to an event at process p

  2. When a process p sends a message m, it timestamps it by including the value t = Cp (after incrementing Cp)

  3. When a process q receives a message (m, t) it sets Cq := max (Cq, t) then Cq is incremented and assigned as a timestamp to the message received event.

- Note: $a \rightarrow b$ implies Ta < Tb but ***not*** Ta < Tb implies $a \rightarrow b$

Logical Clocks - Total Ordering



- Logical Clocks give a partial order on the set of all events as distinct events can have the same identifier.
- A total ordering can be imposed by including the process identifier with the event identifier
- (Ta, Pa) < (Tb, Pb) if and only if Ta < Tb , or Ta = Tb and Pa < Pb
- E.g. $a \rightarrow d$, $d \rightarrow g$, $b \rightarrow h$ using process identifiers

Summary

- Local clock drifting results in non-synchronised clocks

- Synchronisation algorithms have to cope with variable message delays between nodes

- Clock compensation algorithms send local readings, and estimate average delays to derive clock adjustments eg
  - Cristian
  - Berkley
  - NTP

- Logical clocks are sufficient for causal ordering e.g. event dependencies – based on incrementing counters