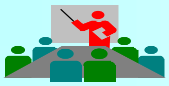
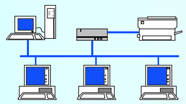




Distributed Systems
600.437
Clock Synchronization

Department of Computer Science
The Johns Hopkins University

Clock Synchronization

Lecture 10

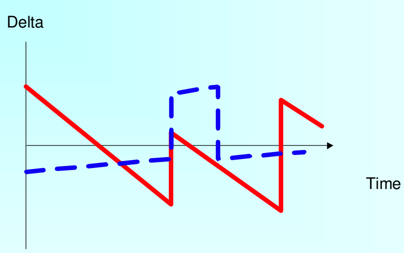
Further reading:

Distributed Operating Systems
Andrew Tanenbaum
Prentice Hall, 1995.

Distributed Systems, Concepts and Design
Coulouris, Dollimore and Kindberg
Addison-Wesley, 1994

Clock Synchronization in the Presence of Omission and Performance Faults, and Processor Joins.
F. Cristian, H. Aghili, R. Strong
In Global States and Time in Distributed Systems, Computer Society Press, pages 69-75, 1993.

The Problem



Clock Drifts of 0.8 seconds/day were measured

Clock Synchronization

Clocks do drift:

- Relatively to each other
  - **Logical clocks** are clocks which are synchronized relatively to each other.
- Relatively to a real world clock
  - How is the real world clock determined?
  - **Physical clocks** are logical clocks which must not deviate from the real time by more than a certain amount.

Synchronizing Logical Clocks

Real time is not important. We care about the ordering of events.

The "happens before" relation => (Lamport):

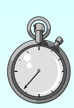
- If *a* and *b* are events in the same process, and *a* occurs before *b*, than *a* => *b*.
- If *a* is the event of a message being sent by one process, and *b* is the event of the message being received by another process than *a* => *b*.

if *a* => *b* then *time*(*a*) < *time*(*b*)

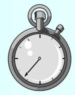
Lamport Logical Clock

P1	P2	P3
0	0	0
4	5	3
8	10	6
12	15	9
16	20	12
20	25	15
24	30	18
28	35	21
32	40	24

P1	P2	P3
0	0	0
4	5	3
8	10	6
12	15	**11**
16	20	14
20	25	17
24	30	20
28	35	23
36	40	26



Lamport Clock (cont)



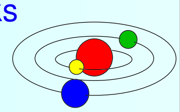
Lamport Clock :

time	proc_id

This way there are no two events which are stamped with the same logical time.

This leads to a total order of events.

Physical Clocks



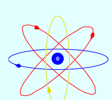
How to measure real time?

- 17th Century: Mechanical Clocks based on astronomy.
- Solar_day = Transit of the Sun (day2 -day1)
- Solar_second = Solar_day / (3600*24)

Problem (1940) : The earth rotation varies (and gets slower and slower).

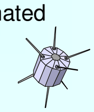
Mean_solar_second : average over many days.

Atomic Clocks



- 1948 - counting transitions of crystal Cesium 133 used as atomic clock.
- TAI = International Atomic time. => 9192631770 transitions = 1 mean solar second at 1948 (but not today...)

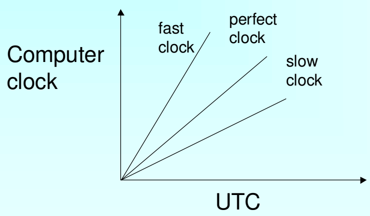
Solution: from time to time (30+ times since 1958) we skip a solar second to stay in phase with the sun. This is UTC (Universal Coordinated Time)



- UTC is broadcast by several sources.

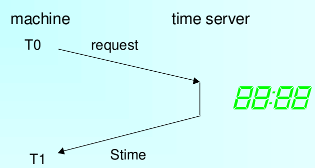
Computer's Clocks are not Accurate

- modern timer chips have a relative error of 1 / 100,000 which means 0.86 second a day!



Cristian's Algorithm

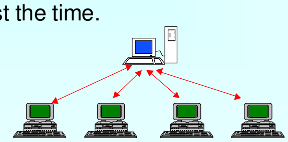
Assuming we have one time server (with UTC).



- Time(T1) is estimated as Stime + (T1-T0)/2.
- Taking the average of several requests is better.
- Taking the estimate of the smallest (T1-T0) might be even better.

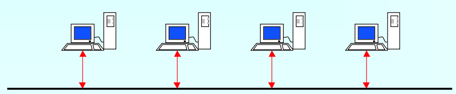
Berkeley Unix Algorithm

- One daemon without UTC
- Periodically, the daemon polls and ask all the machines for their time.
- The machines answer.
- The daemon computes an average time and broadcast the time.



Decentralized Averaging Algorithm

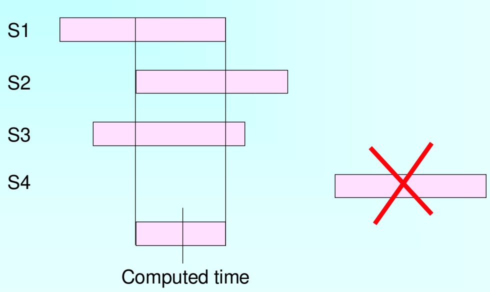
- Each machine has a daemon without UTC
- Periodically, on a fix agreed-upon times, each machine broadcast its local time.
- Each of them calculates the correct time by averaging all the receive points on the local time.



Clock Synchronization in DCE

- DCE's Time model:
  - Time in DCE is actually an **interval.**
  - comparing 2 times may yield 3 answers:
    - t1 < t2
    - t2 < t1
    - not determined.
- Each machine is either a time server or a clerk.
  - Periodically, a clerk contacts ALL the time servers on its LAN
  - Based on their answers it computes a new time and GRADUALLY converges to it.

Clock Synchronization in DCE (cont.)



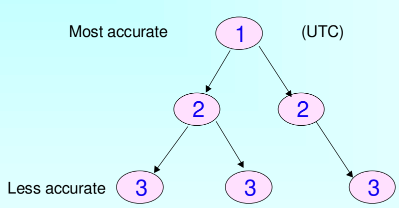
The Network Time Protocol

- Enables clients across the Internet to be synchronized accurately to UTC.
  - Overcomes large and variable message delays.
  - Employ statistical techniques for filtering (based on past quality of servers and several measures).
- Can survive lengthy losses of connectivity.
  - Redundant servers.
  - Redundant paths to a server (using the regular routing protocol)

Network Time Protocol (cont.)

- Enables frequent synchronization.
  - Offsets the drifts found in most computers.
  - Scalable to large networks.
- Provides protection against interference
  - Uses authentication techniques.
- Uses a hierarchy of servers, located across the Internet.
  - The primary servers are directly connected to a UTC time source.

Hierarchy in NTP



Synchronization in NTP

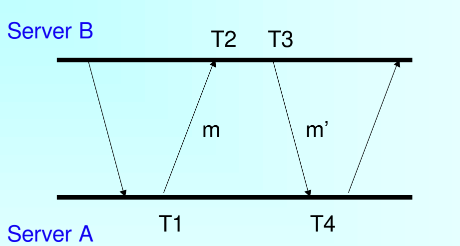
Messages are sent using UDP.

- *Multicast mode*:
  - Suitable for user workstations on a LAN.
  - One or more servers periodically multicast the time to the other machines.
- *Procedure Call mode*:
  - Similar to Cristian's algorithm.
  - Provides higher accuracy compared with the multicast mode, because delays are partially compensated.

Synchronization in NTP (cont.)

- *Symmetric mode*:
  - Used by servers that supply time information to LANs, and by the lower level (more accurate) servers in the hierarchy.
  - The most accurate way, but also the most expensive way.
  - A pair of servers in this mode exchange pairs of timing messages that contain timestamps of recent message events (send and receive).

Symmetric Mode



T2 = T1 + t + offset   T4 = T3 + t' - offset

2 offset -t' + t = T3 - T4 + T2 - T1

t, t' are actual transmission times of m and m'

Calculating Offset and Accuracy

T2 = T1 + t + offset T4 = T3 + t' - offset

2 offset -t' + t = T3 - T4 + T2 - T1

EO = estimated offset = (T3 - T4 + T2 - T1) / 2
accuracy = t + t' ~ T2 - T1 - EO + T4 - T3 + EO
accuracy = T2 - T1 + T4 - T3

Example:

T1 = 5
T2 = 2
T3 = 8
T4 = 13
t, t' = 1

T2 T3
1 2 3 4 5 6 7 8 9

5 6 7 8 9 10 11 12 13
T1 T4

EO = 4
accuracy = 2

Adjusting the time

- Several pairs of (offset estimate, accuracy) are kept for each server that this server communicates with.
- For each such server, the entry with the lowest (best) accuracy is picked.
- Servers that are closer to a UTC are favored.
- Servers which have a smaller sum of accuracy from a root (UTC) are favored.

A Distributed Algorithm

- By Cristian, Aghili & Strong.
- Overcomes message omissions, crashes, performance faults and processor joins.
- Provides approximately synchronized clocks:

$$\forall t : \left| C_p(t) - C_q(t) \right| < DMAX$$

p,q : processors
t: real time
C: local logical time (at a processor)

Assumptions

- **(A1)** The hardware clock HC of a processor drifts in no more than a certain (small) rate.

$$(1+\rho)^{-1} t < HC(t) < (1+\rho) t$$

- **(A2)** let N be the number of max processors. The drift rate is so small that:

$$3N(2\rho + \rho^2) < 1$$

A2 is used only in the performance analysis

Assumptions (cont.)

- **(A3)** A message sent from *p* to *q* (correct processors) on a correct link (p,q) arrives and is processed at q in less than *ldel* real time units (link delay).
- **(A4)** No network partitions: any two correct processes are linked by at least one chain of correct links and intermediate processors.

A3+A4 => *ndel < (N-1)ldel*

ndel : Network delay.

Algorithm's Properties

- **(P1)** There is an a priori known bound for the maximum deviation:

$$\exists DMAX : \forall t : |C_p(t) - C_q(t)| < DMAX$$

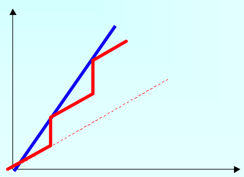
- **(P2)** The clocks of joined correct processors should read times within a linear envelope of real time.

$$X + (1+\beta)^{-1} t < C_p(t) < X' + (1+\beta) t$$

X,X' are constants representing initial conditions

Algorithm's Properties (cont)

- **(P3)** A correct processor that joins a set of synchronized processors should synchronize with them within some a priori know real time delay - *jdel*



Operations

- Operations on logical clocks:
  - **C.adjust(L,T)** adjust the local time L displayed by clock C to T. (can be gradually, over PER time units, or immediate). (PER = clock synchronization period).
  - **C.read** returns the current value of clock c.
- Operations on timers:
  - **TP.set(T)** reset the timer to timeout in T units.
- Message operations:
  - **recv(m,l)** receive a message m at link l.
  - **bcast(m)** send a message m on all links.
  - **forward(m,l)** send message that received at l to all other links.

The Algorithm

Ndel = $(1+\rho)$ndel
Tdel = $2(1+\rho)$ndel

```
Daemon Time-Manager
  s=0, Joined = False
  bcast("new"), TJ.set(NDEL+TDEL)
  do forever
      select event:
          case Timeout TJ:
              joined = True
          case Timeout TP or recv("new",l):
              s++
              L = C.read
              bcast((s,L))
          case recv((s',T),l)
              *** next slide ***
      TP.set(PER)
  end loop
```

The Algorithm (cont.)

```
case recv((s',T),l):
        L= C.read
        if (s'<s) or (s'=s & T <= L)
                continue to loop (without setting TP);
        if (s'=s) &(T>L)
                C.adjust(L,T); forward( (s,T), l);
        if (s'>s) &(T<L)
                s=s'; bcast((s,L))          --I am faster
        if (s'>s) &(T>=L)
                s=s'; C.adjust( L,T ); forward( (s,T),l)
                        -- synchronizing my time.
```