CSCI 5673
Distributed Systems

Lecture Set Two

Clock synchronization

Lecture Notes by
Shivakant Mishra
Computer Science, CU Boulder
Last update: January 14, 2017

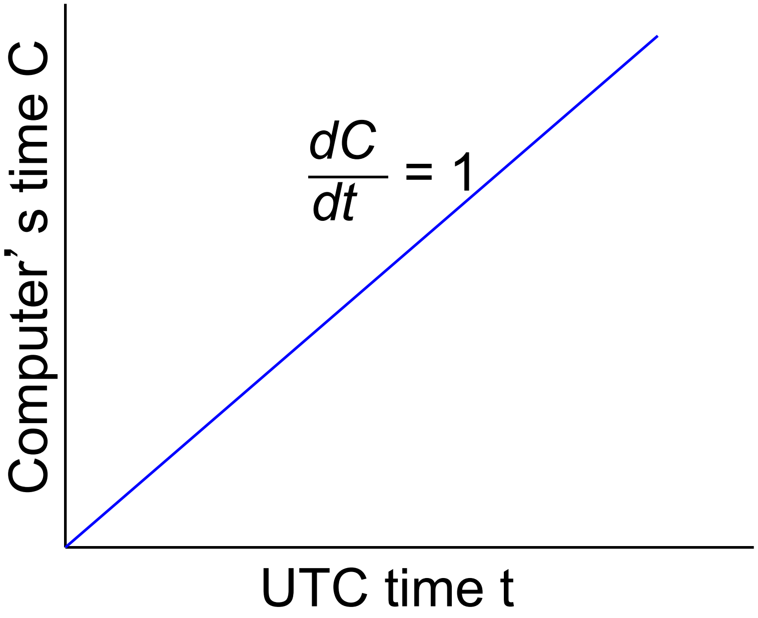
Physical clocks in computers

- CMOS clock (counter) circuit driven by a quartz oscillator
  - Quartz crystal creates an electric signal with a very precise frequency
  - Battery backup to continue measuring time when power is off
- OS programs a timer circuit to generate an interrupt periodically
  - e.g., 60 or 100 interrupts per second
  - Programmable Interrupt Controller (PIC)
  - Interrupt service routine adds a fixed number to a counter in memory

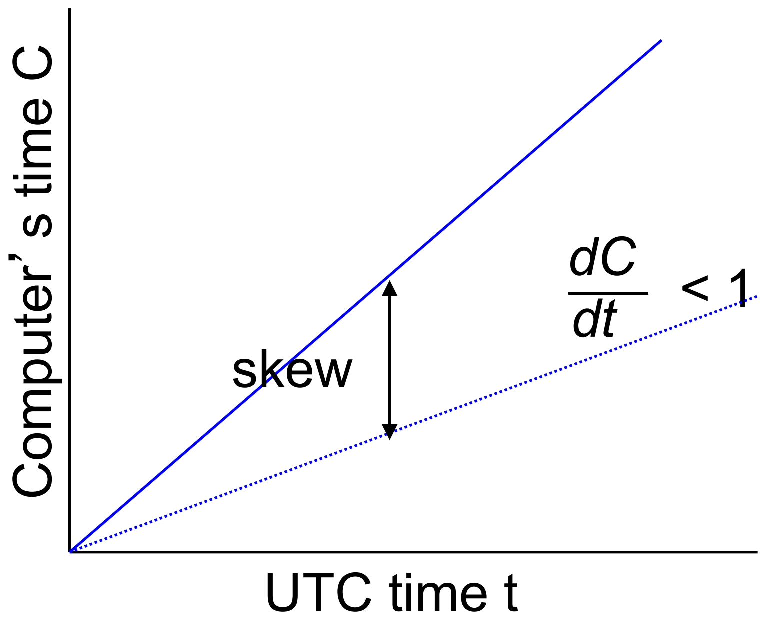
Problem

- Getting two systems to agree on time
  - Two clocks hardly ever agree
  - Quartz oscillators oscillate at slightly different frequencies
- Clocks tick at different rates
  - Create ever-widening gap in perceived time
  - Clock Drift
- Difference between two clocks at one point in time
  - Clock Skew

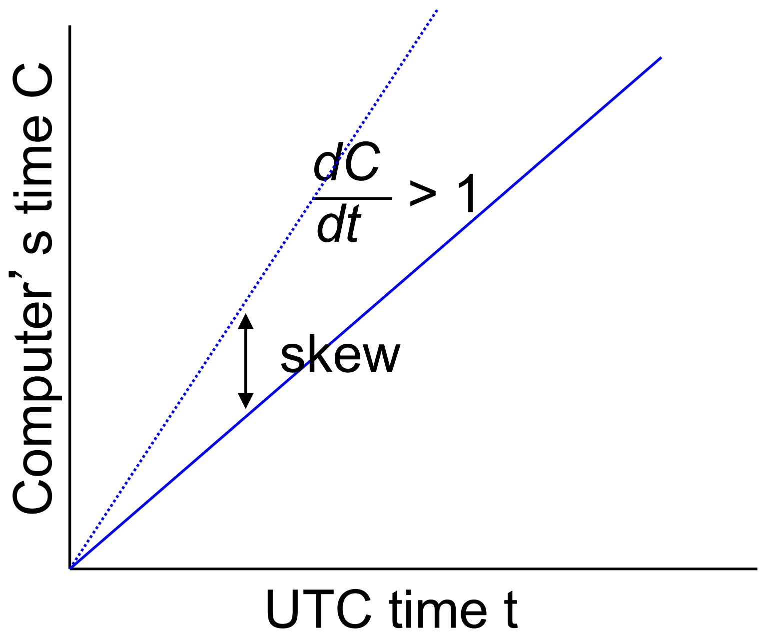
Perfect clock



Drift with slow clock



Drift with fast clock



Dealing with drift

- Set computer to true time at regular intervals
  - Not a good idea to set clock back
    - Illusion of time moving backwards
    - Can confuse message ordering and software development environments

Dealing with drift

Go for *gradual* clock correction

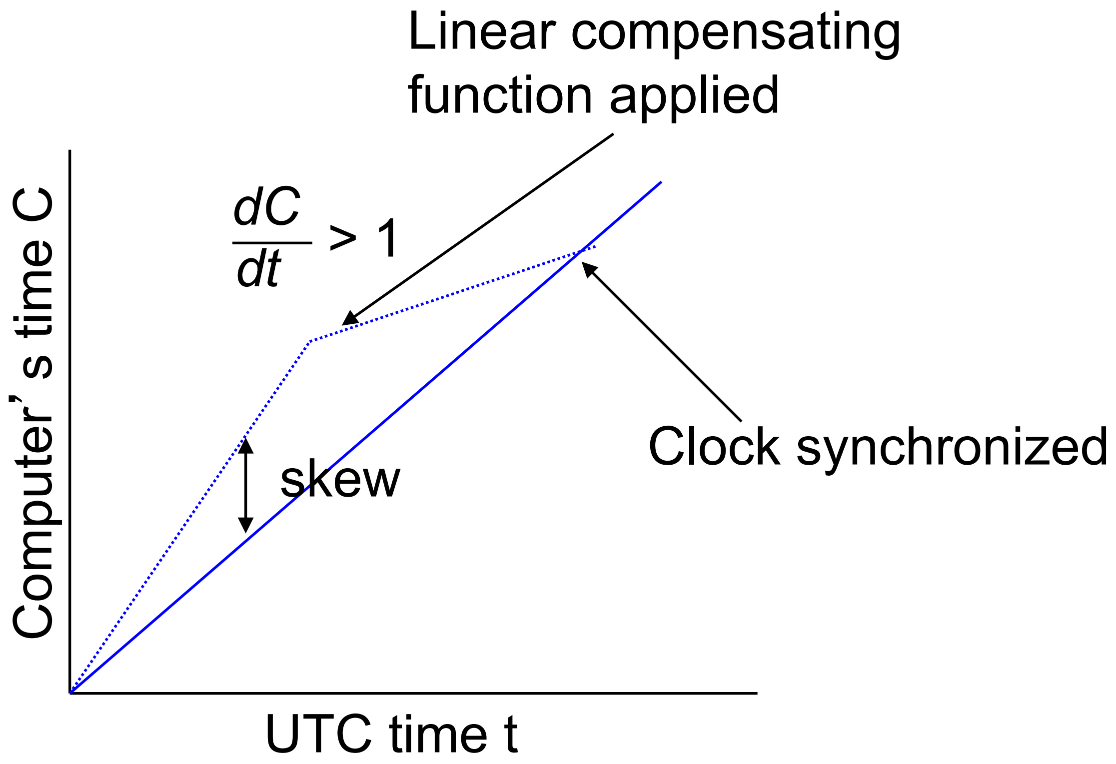
- If fast:
  - Make clock run slower until it synchronizes
- If slow:
  - Make clock run faster until it synchronizes

Dealing with drift

OS can do this:

- Change the update value of the counter
  - e.g. If ISR adds 1000 on an interrupt, but the clock is too slow:
    - Add 1200
- Adjustment changes slope of system time
  - Linear compensating function

Compensating for a fast clock



Resynchronizing

- After synchronization period is reached
  - Resynchronize periodically
  - Successive application of a second linear compensating function can bring us closer to true slope
- Keep track of adjustments and apply continuously
  - e.g., UNIX *adjtimex( )* system call

UTC

- Coordinated Universal Time: Time standard by which the world regulates clocks and time.
  - roughly equal to Greenwich Mean Time (GMT)
- Derived from two sources:
  - International Atomic Time (TAI): A time scale that combines the output of some 400 highly precise atomic clocks worldwide
  - Universal Time (UT1): also known as astronomical time or solar time, refers to the Earth's rotation. It is used to compare the pace provided by TAI with the actual length of a day on Earth

Getting accurate time

- Attach GPS receiver to each computer
  - ± 1 msec of UTC
- Attach WWV radio receiver
  - Obtain time broadcasts from Boulder or DC
  - ± 3 msec of UTC (depending on distance)
- Not a practical solution for every machine
  - Cost, size, convenience, environment

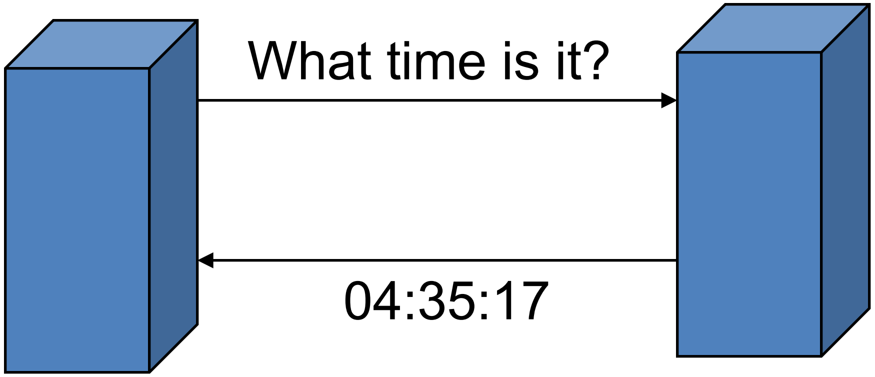
Getting accurate time

- Synchronize from another machine
  - One with a more accurate clock
- Machine that provides time information

  Time server

RPC

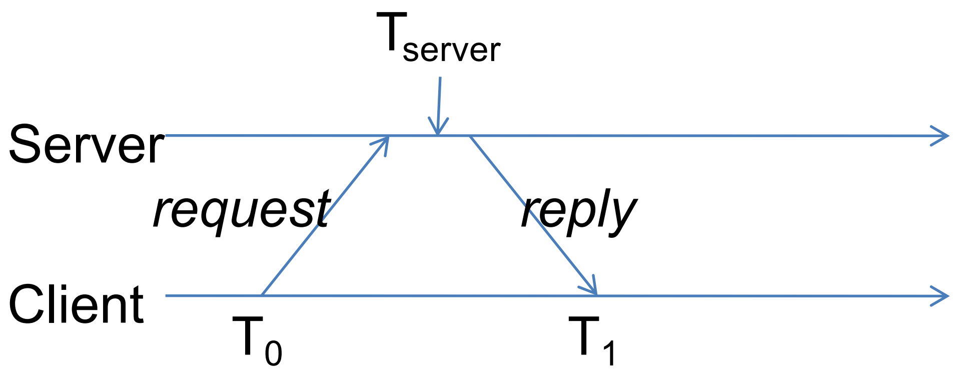
- Simplest synchronization technique
  - Issue RPC to obtain time
  - Set time

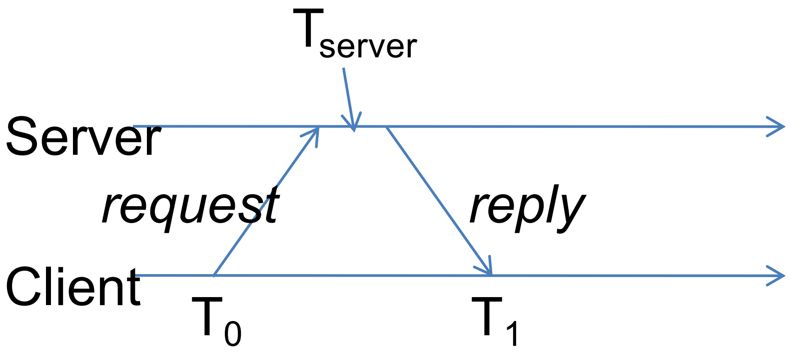


Does not account for network or processing latency

Clock synchronization: Algorithm 1

- Reference: F. Cristian. *Probabilistic Clock Synchronization,* Distributed Computing, 3, 1989.
- Compensate for delays
  - Note times:
    - request sent: T0
    - reply received: T1
  - Assume network delays are symmetric





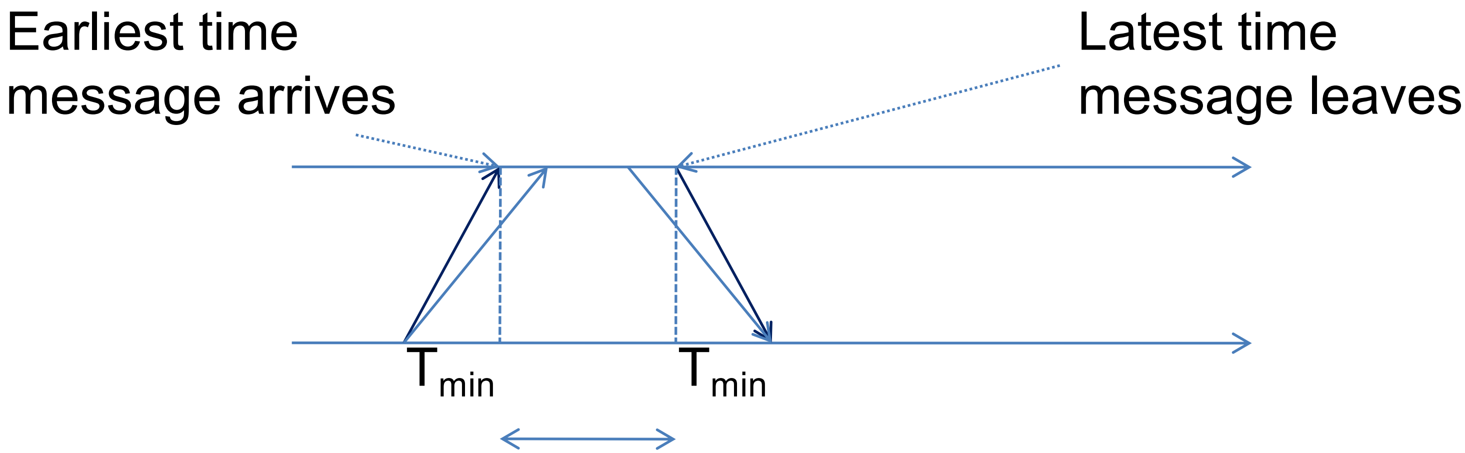
Client sets time to $T_{new} = T_{server} + \dfrac{T_1 - T_0}{2}$

Error bounds: $\pm \dfrac{T_1 - T_0}{2}$

Error bounds

- If minimum message transit time (T_{min}) *is* known
  - Place bounds on accuracy of result

Error bounds



$$\text{Range} = T_1 - T_0 - 2T_{min}$$

$$\text{Error bounds: } \pm \left(\frac{T_1 - T_0}{2} - T_{min}\right)$$

Clock Synchronization: Algorithm 2

- Reference: Gusella & Zatti. *The Accuracy of the Clock Synchronization Achieved by TEMPO in Berkeley UNIX 4.3BSD,* IEEE Transactions on Software Engineering, 15(7),1989.
  - Assumes no machine has an accurate time source
  - Obtains average from participating computers
  - Synchronizes all clocks to average

Clock Synchronization: Algorithm 2

- Machines run time dæmon
  - Process that implements protocol
- One machine is elected (or designated) as the server (master)

  Others are slaves

Clock Synchronization: Algorithm 2

- Master polls each machine periodically
  - Ask each machine for time
  - Can use algorithm 1 to compensate for network latency
- When results are in, compute average
  - Including master's time
- *Hope: average cancels out individual clock's tendencies to run fast or slow*
- Send offset by which each clock needs adjustment to each slave
  - Avoids problems with network delays if we send a time stamp

Clock Synchronization: Algorithm 2

- Algorithm has provisions for ignoring readings from clocks whose skew is too great
  - Compute a fault-tolerant average
- If master fails
  - Any slave can take over

Network Time Protocol, NTP

- NTP is a distributed time synchronization protocol
  - Networking protocol for clock synchronization over packet-switched, variable-latency data networks
  - Keeps computers synchronized to UTC despite large and variable delays of Internet
- 1991, 1992
- Internet Standard, version 4: RFC 5905
- *http://www.ntp.org/*
- K. A. Marzullo. Maintaining the Time in a Distributed System: An Example of a Loosely-Coupled Distributed Service. Ph.D. dissertation, Stanford University, Department of Electrical Engineering, February 1984

NTP Goals

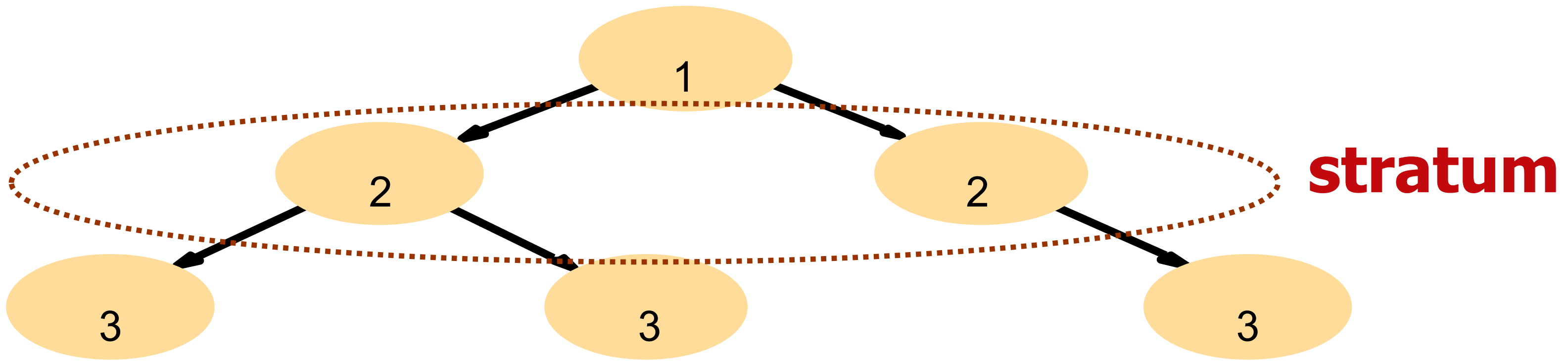
- Enable clients across Internet to be accurately synchronized to UTC despite message delays
  - Use statistical techniques to filter data and gauge quality of results
- Provide reliable service
  - Survive lengthy losses of connectivity
  - Redundant paths
  - Redundant servers
- Enable clients to synchronize frequently
  - Offset effects of clock drift
- Provide protection against interference
  - Authenticate source of data

NTP

- Statistical filtering of timing data
  – Discrimination based on quality of data from different servers
- Re-configurable inter-server connections
  – Logical hierarchy
- Scalable for both clients & servers
  – Clients can re-sync; frequently to offset drift
- Authentication of trusted servers
  – … and also validation of return addresses

<u>Sync. Accuracy</u>: ~10s of milliseconds over Internet paths
~ 1 millisecond on LANs

NTP servers



- Arranged in strata
  - 1st stratum: machines connected directly to accurate time source →atomic clocks, GPS clocks, radio clocks, … (with in few micro sec of stratum 0 --- high precision time keeping devices)
  - 2nd stratum: machines synchronized from 1st stratum machines; peer with other stratum 2 servers
  - …

Larger stratum # → server more liable to be less accurate

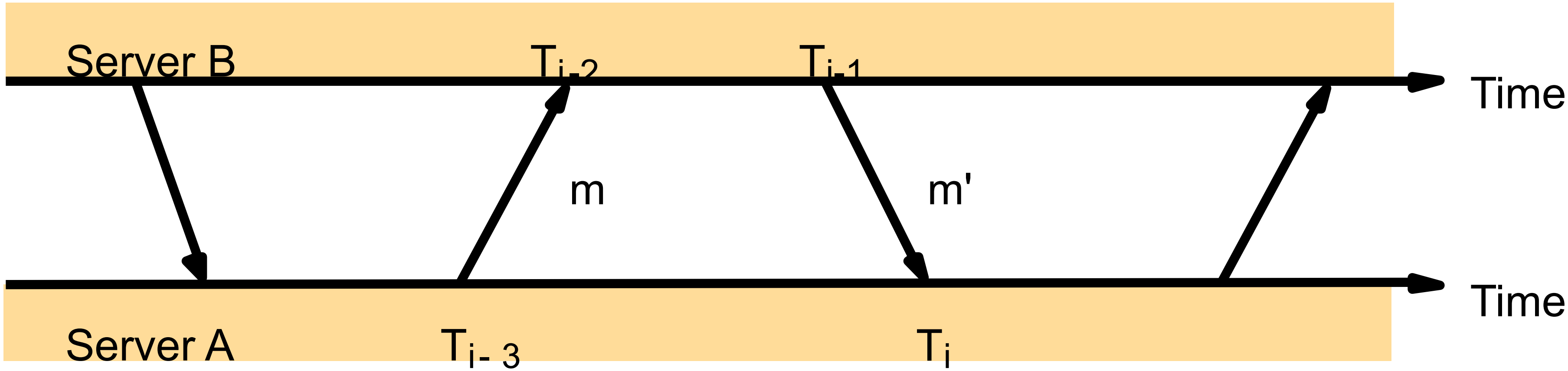
NTP Synchronization Modes

- Multicast mode
  - High speed LANS
  - Lower accuracy but efficient
- Procedure call mode
  - Similar to Cristian's algorithm (Algorithm 1)
- Symmetric mode
  - Intended for master servers
  - Pair of servers exchange messages and retain data to improve synchronization over time

All messages delivered unreliably with UDP

NTP messages

- Procedure call and symmetric mode
  - Messages exchanged in pairs
- NTP calculates:
  - Offset for each pair of messages
    - Estimate of offset between two clocks
  - Delay
    - Transmit time between two messages
  - Filter Dispersion
    - Estimate error – quality of results
    - Based on accuracy of server's clock *and* consistency of network transit time
- Use this data to find preferred server:
  - *lower stratum & lowest total dispersion*



Each message contains the local times when the previous message was sent & received, and the local time when the current message was sent.

- There can be a non-negligible delay between the arrival of one message & the dispatch of the next.
- Messages may be lost

Offset o_i : estimate of the actual offset between two clocks as computed from a pair of messages

Delay d_i : total transmission time for the message pair

$$T_{i-2} = T_{i-3} + t + o, \text{ where } o \text{ is the true offset}$$

$$T_i = T_{i-1} + t' - o$$

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

$$o = (T_{i-2} - T_{i-3} - T_i + T_{i-1}) / 2 + (t' - t)/2$$

$$o_i = (T_{i-2} - T_{i-3} - T_i + T_{i-1}) / 2$$

$$\boxed{o_i - d_i/2 <= o < o_i + d_i/2}$$

- Delay d_i is a measure of the accuracy of the estimate of offset
- The shorter and more symmetric the round-trip time, the more accurate the estimate of the current time

NTP data filtering & peer selection

- Retain 8 most recent <o_i, d_i> pairs
  - Compute "filter dispersion" metric
    - higher values → less reliable data
    - The estimate of offset with min. delay is chosen
- Examine values from several peers
  - look for relatively unreliable values
- May switch the peer used primarily for sync.
- Peers with low stratum # are more favored
  - "closer" to primary time sources
- Also favored are peers with lowest sync. dispersion
  - sum of filter dispersions bet. peer & root of sync. subnet
- May modify local clock update frequency wrt observed drift rate

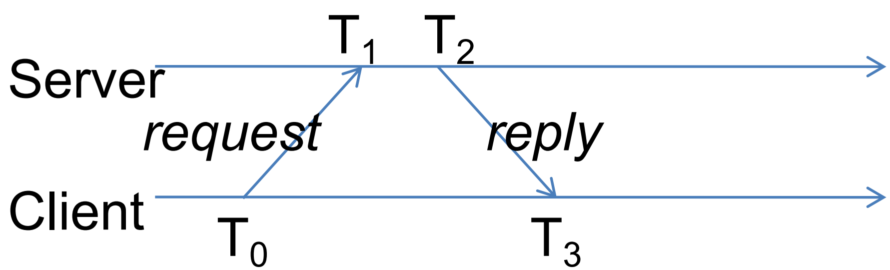
NTP Software Implementation

- Unix: the NTP client is implemented as a daemon process *ntpd* that runs continuously in user space
- Windows: Windows Time Service ("w32time")
- Linux: *ntimed* (No official release yet; started in 2014)
- Timestamp
  - 64 bits: 32 bit second, 32 bit fractional second
  - theoretical resolution of 2^{-32} seconds (233 picoseconds)
  - Epoch of January 1, 1900 →rollover in 2036
  - Future versions: may extend to 128 bits

SNTP

- Simple Network Time Protocol
  - Based on Unicast mode of NTP
  - Subset of NTP, not new protocol
  - Operates in multicast or procedure call mode
  - Recommended for environments where server is root node and client is leaf of synchronization subnet
  - Root delay, root dispersion, reference timestamp ignored, no storage of state over extended periods of time
  - Used in some embedded devices
- RFC 2030, October 1996

SNTP



Roundtrip delay: $d = (T_3 - T_0) - (T_2 - T_1)$

Time offset: $t = \dfrac{(T_1 - T_0) + (T_2 - T_3)}{2}$