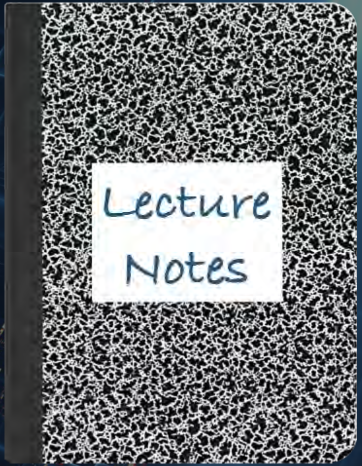
CS 417 – DISTRIBUTED SYSTEMS

Week 3: Part 1

Clock synchronization



Paul Krzyzanowski

Synchronization

Synchronization covers interactions among distributed processes

Clocks	Identify *when* something happened
Mutual exclusion	Only one entity can do an operation at a time
Leader election	Who coordinates activity?
Message consistency	Does everyone have the same view of events?
Agreement	Can everyone agree on a proposed value?

All of these are trivial in non-distributed systems

All of these are tricky in distributed systems

Clock Synchronization

Why?

- Allow a process to identify "*now*" in a way that's consistent with other processes on other systems
  – Scheduling jobs in process control environments
  – Applications where time-based billing or access control is needed
  – Certificate validation

- Temporal ordering of events from concurrent processes
  – Event logging: debugging, root cause analysis, tracking breaches
  – Consistent file modification times in shared file systems
  – Identifying latest versions

Logical vs. physical clocks

- Physical clocks keep time of day
  - Consistent across systems
- Logical clock keeps track of event ordering
  - among related (causal) events

Synchronizing physical clocks

Problem: Get two systems to agree on time

- Why is it hard?
  - 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
- Relative offset = Difference between two clocks at one point in time
- Jitter = Short-term variation in frequency
- Also note — astronomical time vs. relative time
  - Time of day vs. count of seconds from epoch

    (e.g., Unix/Linux counts seconds from 00:00:00 UTC on 1 January 1970)
  - Time of day takes time zones, daylight saving time, leap seconds, etc. into account

Dealing with drift

Not good idea to set a clock back

- Illusion of time moving backwards can confuse message ordering and software development environments

Apply *gradual* clock correction

If fast (ahead):

Make the clock run slower until it synchronizes

If slow (behind):

Make the clock run faster until it synchronizes

Dealing with drift

The OS can do this:

1. Redefine the rate at which system time is advanced with each interrupt

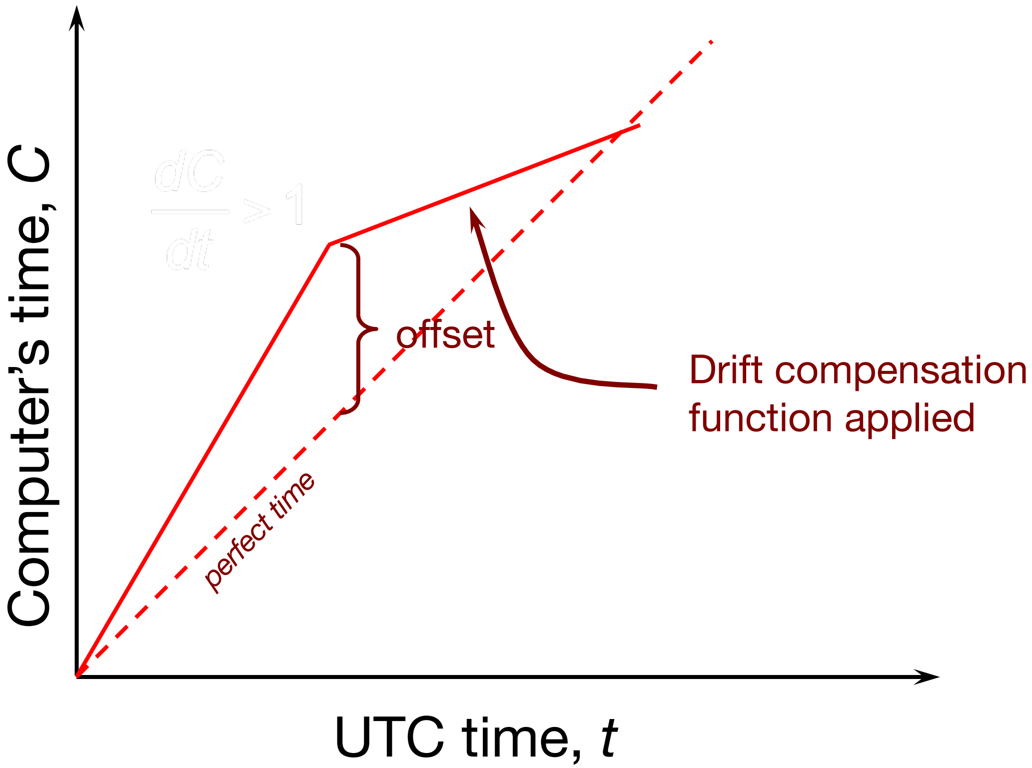
or

2. Read the counter but compensate for drift

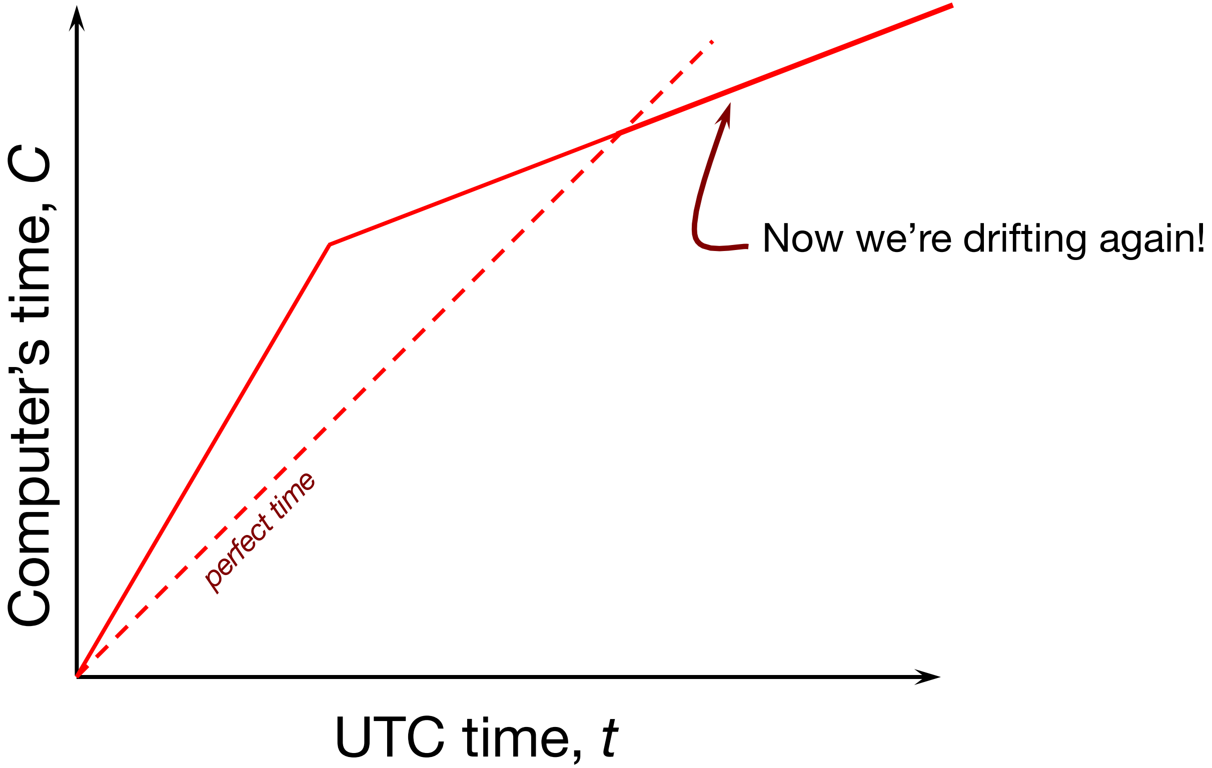
Adjustment changes slope of system time:

Drift compensation via a linear compensation function

Compensating for a fast clock



Compensating for a fast clock



Resynchronizing

After synchronization period is reached

- Resynchronize periodically
- Successive adjustment of a drift compensation function can bring us closer to true slope

Long-term clock stability is not guaranteed

The system clock will still drift based on changes in temperature, pressure, humidity, and age of the crystal

Keep track of adjustments and apply continuously

- e.g., Linux *adjtimex* system calls and *hwclock* command

Going to sleep

- RTC keeps on ticking when the system is off (or sleeping)
- OS cannot apply correction continually
- Record time when going to sleep
  - Read hardware clock on wake-up
  - Estimate drift for the interval and apply a correction factor

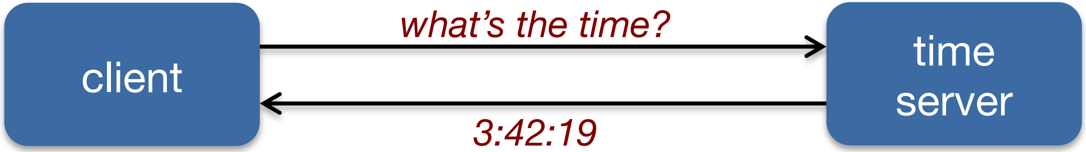
Getting accurate time

- Attach GPS receiver to each computer
  - Accurate to ~40 ns
- Not a practical solution for every machine
  - Cost, power, convenience, environment
  - Accuracy gets worse near buildings, bridges, trees, …
- Chip-scale atomic clock
  - Nice, but around $2,000
  - Most computers won't have this either
  - And if you have it, you still need to set it to give you the right time

Synchronize from a time server

Simplest synchronization technique

- Send a network request to obtain the time
- Set the time to the returned value

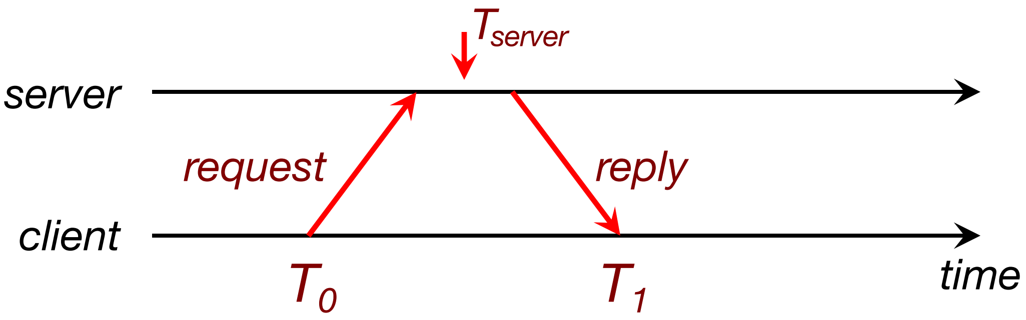


Does not account for network or processing latency

Cristian's algorithm

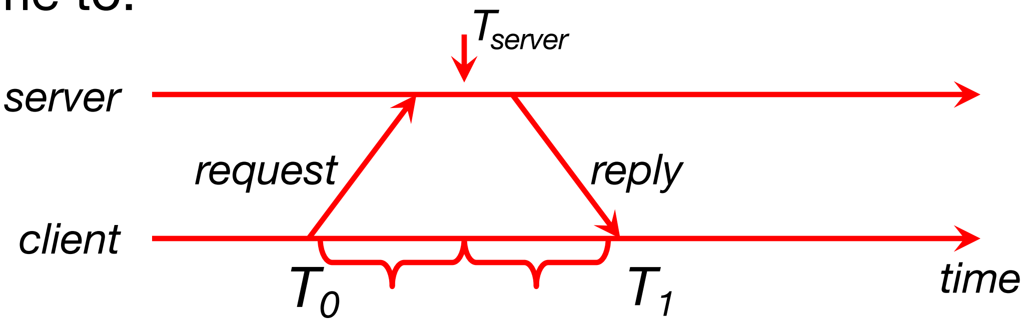
Compensate for delays

- Note times:
  - request sent: T_0
  - reply received: T_1
- Assume network delays are symmetric



Cristian's algorithm

Client sets time to:



$$\frac{T_1 - T_0}{2} =$$ estimated overhead in each direction

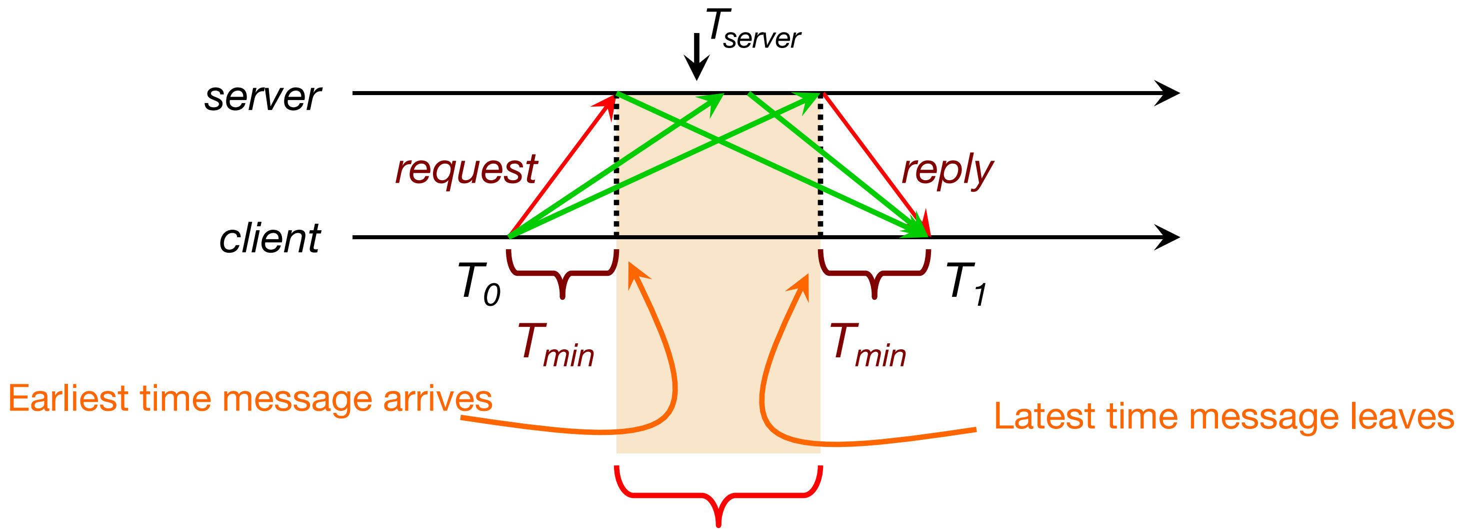
$$T_{new} = T_{server} + \frac{T_1 - T_0}{2}$$

Error bounds

If the minimum message transit time (T_{min}) is known:

Place bounds on accuracy of result

Error bounds



range = $T_1 - T_0 - 2T_{min}$

$$\boxed{\text{accuracy of result} = \pm\frac{T_1 - T_0}{2} - T_{min}}$$

Cristian's algorithm: example

- Send request at 5:08:15.100 (T_0)
- Receive response at 5:08:15.900 (T_1)
- Response contains 5:09:25.300 (T_{server})

Note:
1,000 ms = 1 s
1,000,000 μs = 1s

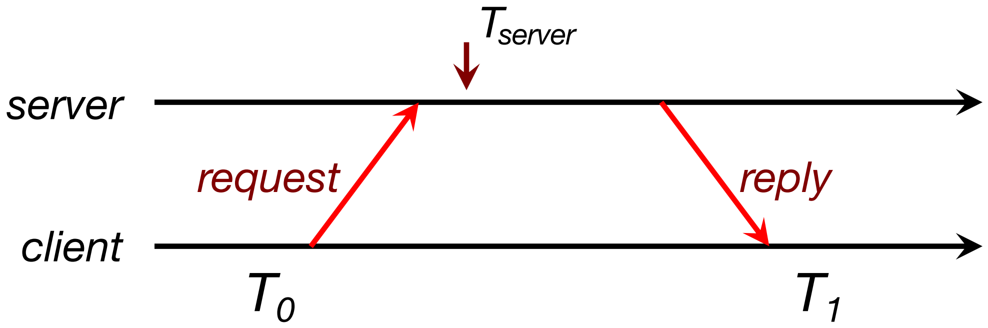
Elapsed time is $T_1 - T_0$ = 5:08:15.900 - 5:08:15.100 = 800 ms

Best guess: timestamp was generated 400 ms ago

Set time to T_{server}+ *elapsed time* = 5:09:25.300 + 0.400 = 5:09.25.700

Cristian's algorithm: example

If best-case message time=200 ms



T_0 = 5:08:15.100

T_1 = 5:08:15.900

T_s = 5:09:25:300

T_{min} = 200 ms

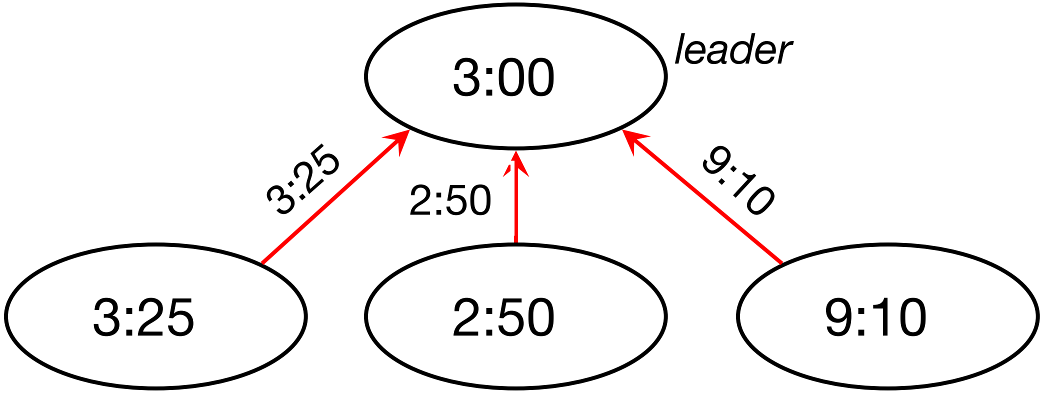
$$\text{Error} = \pm\frac{900-100}{2} - 200 = \pm\frac{800}{2} - 200 = \pm 200\ ms$$

Berkeley Algorithm

Gusella & Zatti, 1989

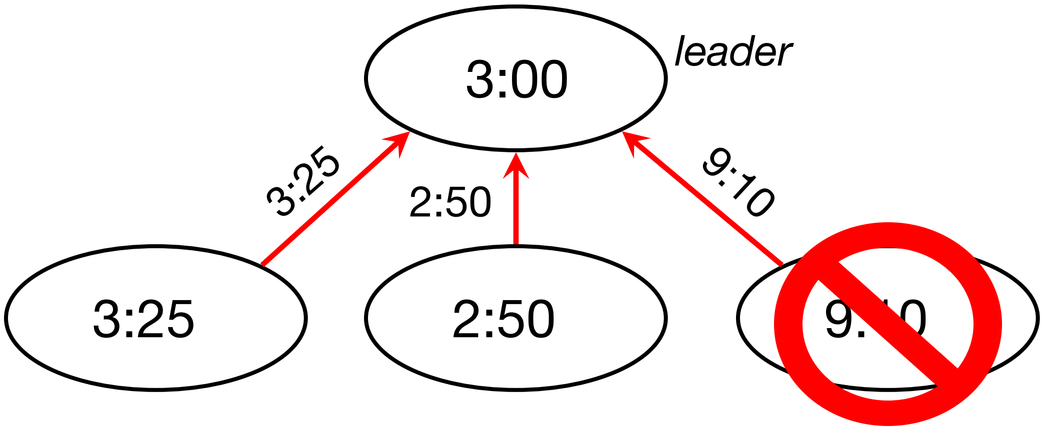
- Designed for intranets (e.g., data centers)
- Assumes no machine has an accurate time source
- Obtains time from participating computers
- Synchronizes all clocks to a fault-tolerant average
  - Select the largest set of time values that don’t differ from each other by some quantity
  - Avoids averaging values of malfunctioning clocks or clocks that drifted too far

Berkeley Algorithm: example



1. Request timestamps from all followers

Berkeley Algorithm: example

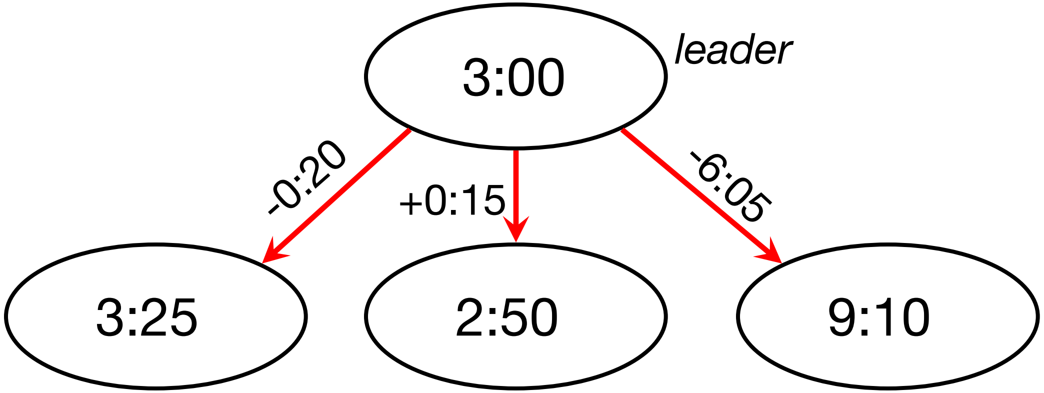


Suppose max ∂=0:45

2. Compute fault-tolerant average:

$$\frac{3:25 + 2:50 + 3:00}{3} = 3:05$$

Berkeley Algorithm: example



3. Send offset to each client

Network Time Protocol, NTP

- 1991, 1992
  - Internet Standard, version 3: RFC 1305
- June 2010
  - Internet Standard, version 4: RFC 5905-5908
  - IPv6 support
  - Improve accuracy to tens of microseconds
  - Dynamic server discovery
- November 2020
  - Draft standard, NTP v5

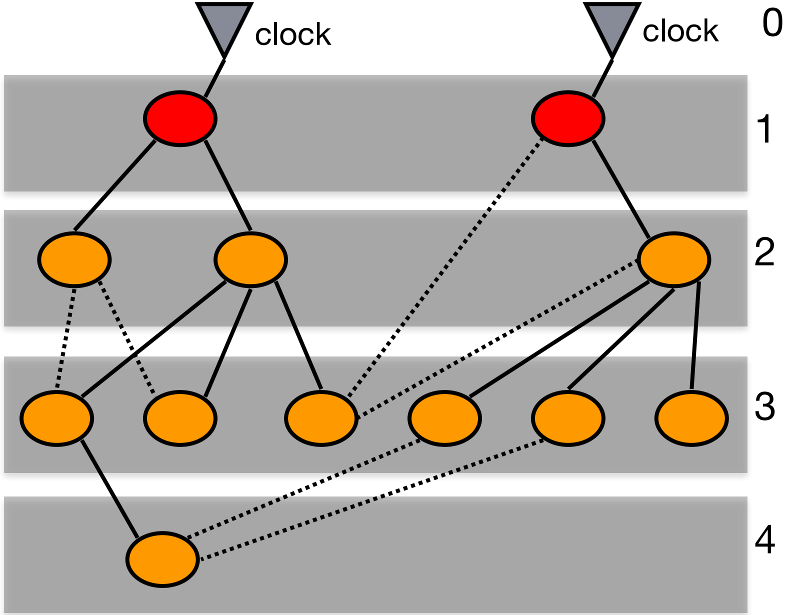
NTP Goals

- 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
- Provide scalable service
  - Enable huge numbers of clients to synchronize frequently
  - Offset effects of clock drift
- Provide protection against interference
  - Authenticate source of data

NTP servers

Arranged in ***strata***

- **Stratum 0** = master clock
- **Stratum 1**: systems connected directly to accurate time source
- **Stratum 2**: systems synchronized from 1st stratum systems
- …
- **Stratum 15**: systems synchronized from 14th stratum systems



Synchronization Subnet

NTP Synchronization Modes

Broadcast (or multicast) mode

Not supported in NTPv5

- Lower accuracy but efficient; for high-speed LANs

Client-server (procedure call) mode

- Cristian's algorithm

Symmetric (peer-to-peer) mode

Not supported in NTPv5

- Peer servers can synchronize with each other to provide mutual backup
  - Usually used with stratum 1 & 2 servers
  - Pair of servers retain data to improve synchronization over time
  - Both devices act as requesters and responders – they operate in the same stratum and the times converge to each other

All messages are delivered unreliably with UDP (port 123)

NTP Clock Quality

- Precision
  - Smallest increment of time that can be read from the clock
- Jitter (dispersion)
  - Difference in successive measurements
  - Due to network delays, OS delays, and clock oscillator instability
- Accuracy
  - How close is the clock to UTC?

NTP messages

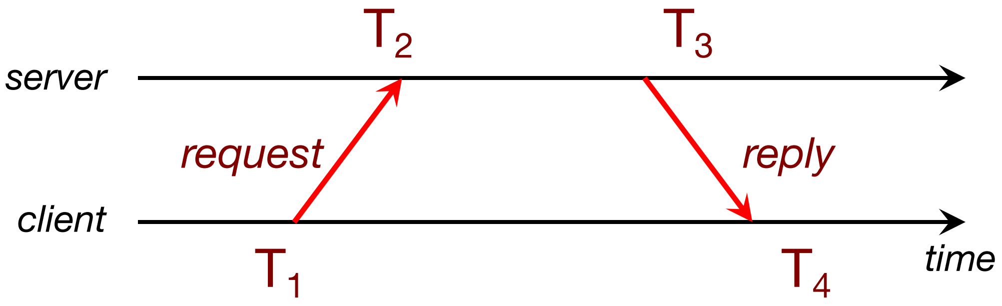
- Procedure call and symmetric mode
  - Messages exchanged in pairs: request & response
- Time encoded as a 64-bit value:
  - Divide by 2^{32} to get the number of seconds since Jan 1 1900 UTC
- NTP calculates:
  - Offset for each pair of messages (θ): Estimate of time difference between two clocks
    = correction that needs to be applied to a client clock to synchronize it
  - Delay (δ): Travel time: ½ of total delay minus remote processing time
  - Dispersion = jitter: Maximum offset error relative to reference clock
    - Found via repeated synchronizations
- Use this data to find the preferred NTP server:
  - Probe multiple servers – each several times
  - ***Pick lowest dispersion … at the lowest stratum if tied***

Simple Network Time Protocol (SNTP)

- 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
- Does not
  - Account for servers that provide inconsistent time stamps
  - Detect malicious interference

v3 RFC 2030, October 1996; v4 RFC 5905, June 2010

SNTP Example



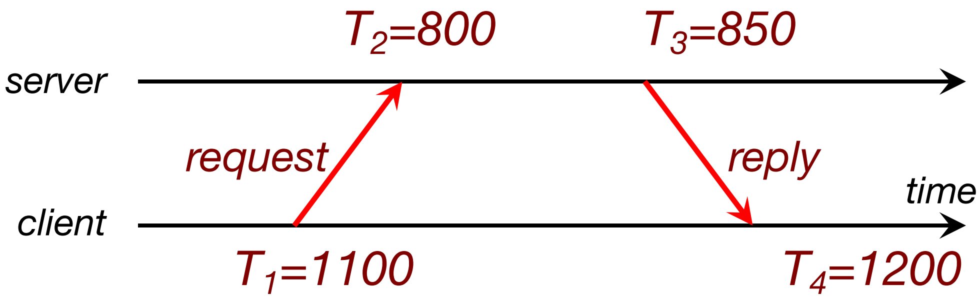
Round-trip network delay:

$$\partial = (T_4 - T_1) - (T_3 - T_2)$$

Time offset:

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

SNTP example



Offset = ((800 - 1100) + (850 - 1200)) / 2

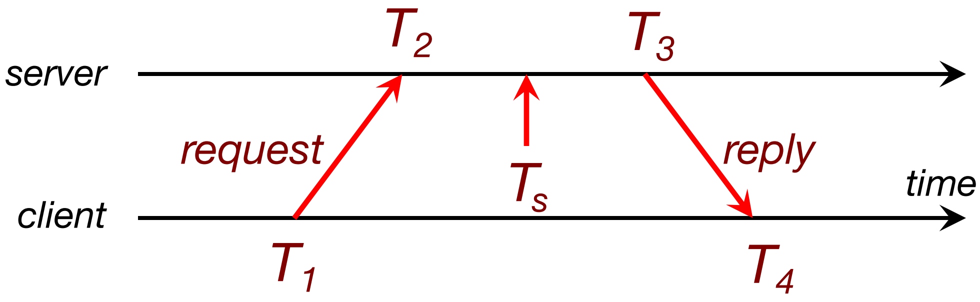
= ((-300) + (-350)) / 2

= -650 / 2 = **-325**

Time offset: $t = \dfrac{(T_2 - T_1) + (T_3 - T_4)}{2}$

Set time to $T_4 + t$ = 1200 - 325 = 875

SNTP = Cristian's algorithm



Just define $T_s = \dfrac{T_2 + T_3}{2}$

Key Points: Physical Clocks

- Cristian's algorithm & SNTP
  - Set clock from server
  - But account for network delays
  - Error: uncertainty due to network/processor latency
    - Errors are additive
    - Example: ±10 ms and ±20 ms = ±30 ms
  - NTP: track jitter, error, stratum, and delay among several NTP servers to choose the best one to synchronize from
- Adjust for local clock drift
  - Linear compensation function

Precision Time Protocol

More accurate clock synchronization

- Why?
  - Industrial process control: synchronizing equipment
  - High frequency trading
  - Power-grid controls
  - Audio/video sync

PTP: IEEE 1588 Precision Time Protocol

- Designed to synchronize clocks on a LAN to sub-microsecond precision
  - Designed for LANs, not global: low jitter, low latency
  - Timestamps ideally generated at the MAC or PHY layers to minimize delay and jitter

- Determine master clock (called the Grandmaster)
  - Use a Best Master Clock algorithm to determine which clock is most precise
  - The Grandmaster sends periodic synchronization messages to others (slave devices)

- Two phases in synchronization
  1. Offset correction
  2. Delay correction

PTP: Choose the "best" clock

Best Master Clock

- Distributed election based on properties of clocks
- Criteria from highest to lowest:
  - Priority 1 (admin-defined hint)
  - Clock class
  - Clock accuracy
  - Clock variance: estimate of stability based on past syncs
  - Priority 2 (admin-defined hint #2)
  - Unique ID (tie-breaker)

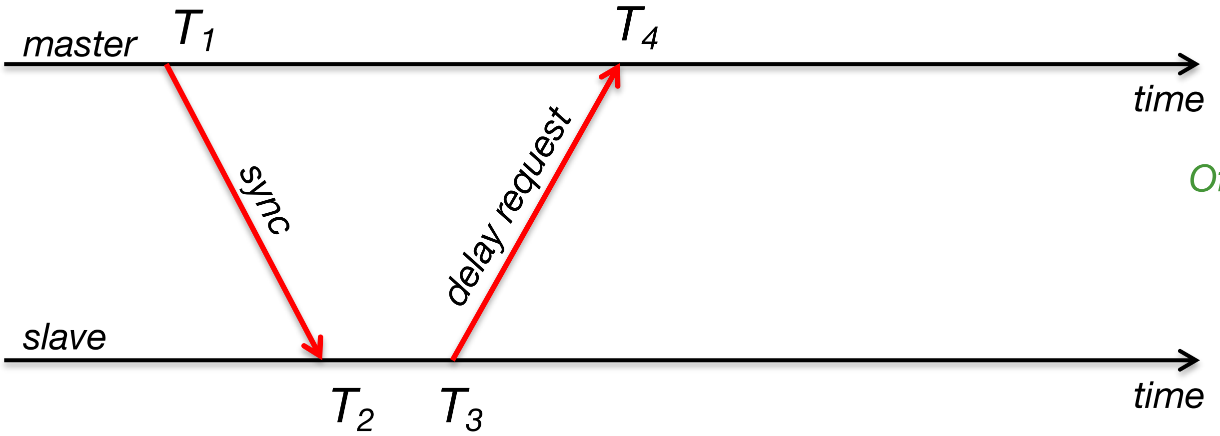
PTP: Master initiates sync

Master initiates the protocol by sending a *sync* message containing a timestamp

Slave timestamps arrival with a timestamp from its local clock

Offset + Delay = $T_2 - T_1$

PTP: Send delay request

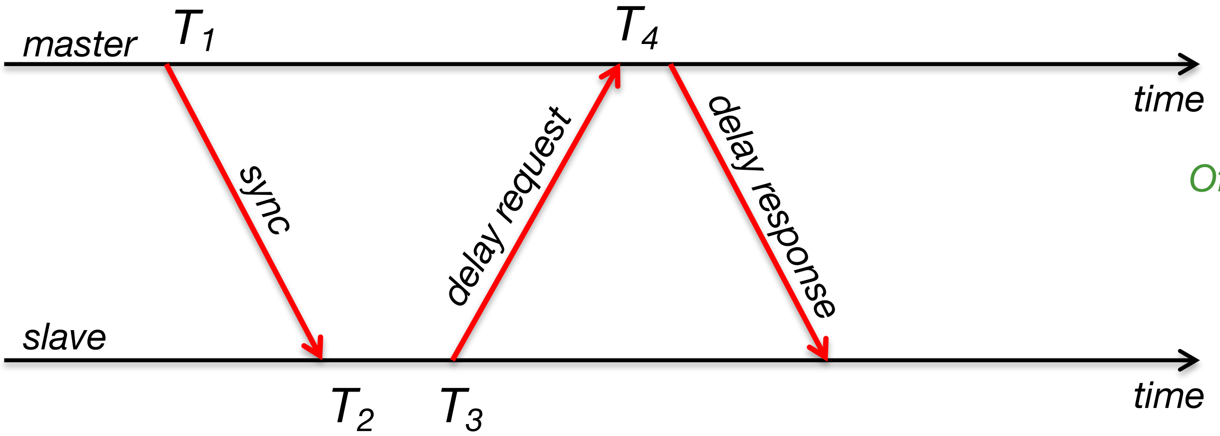


Offset + Delay = T_2 - T_1

Slave needs to figure out the network delay. Send a *delay request*

Note the time it was sent

PTP: Receive delay response

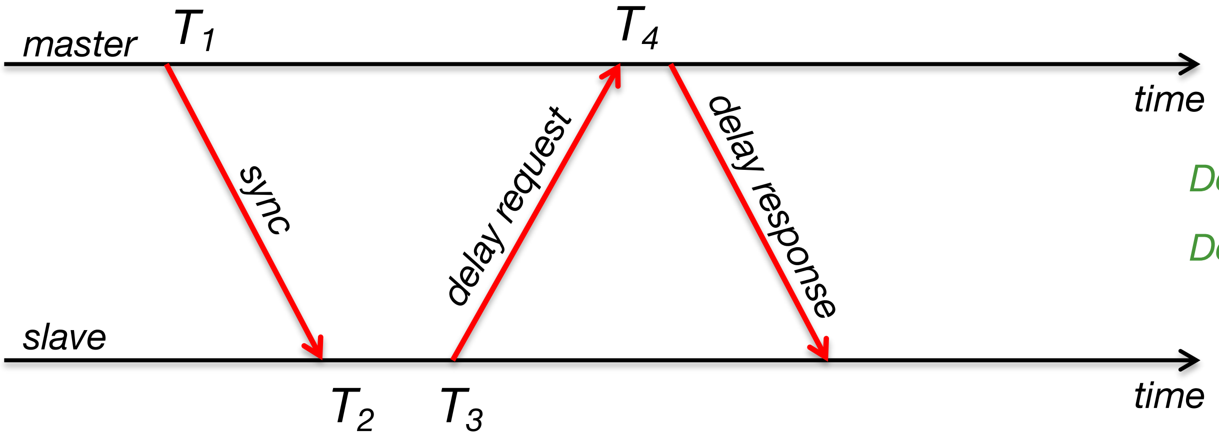


Offset + Delay = T_2 - T_1

Master marks the time of arrival and returns it in a *delay response*

$$\textit{Delay response} = \textit{Delay} - \textit{Offset} = T_4 - T_3$$

PTP: Slave computes offset



Delay + Offset = T_2 - T_1

Delay - Offset = T_4 - T_3

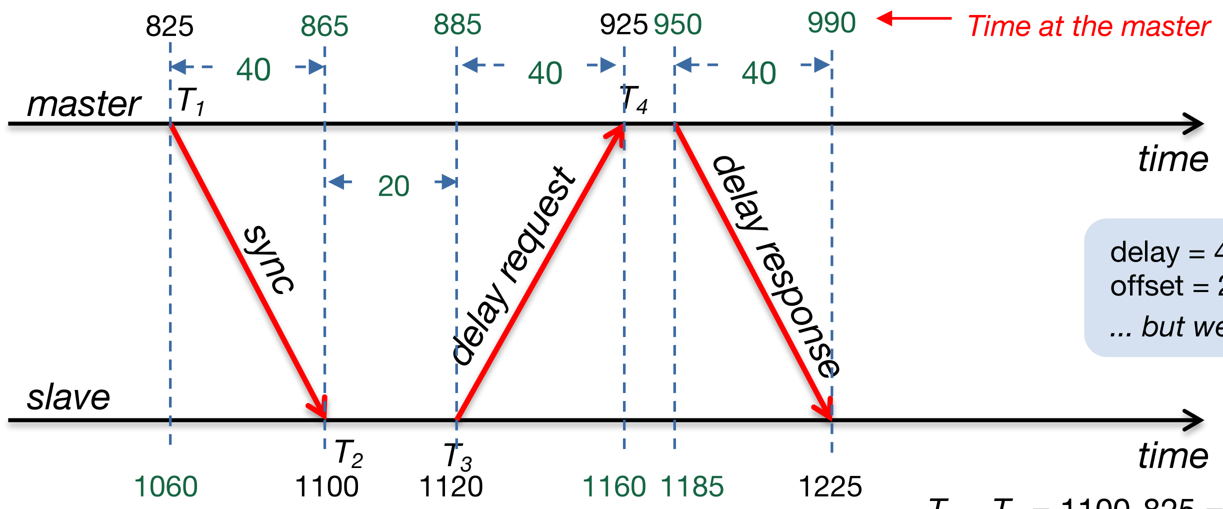
$$\begin{array}{rl} & master\_slave\_difference = T_2 - T_1 = delay + offset \\ - & slave\_master\_difference = T_4 - T_3 = delay - offset \\ \hline \end{array}$$

$$master\_slave\_difference - slave\_master\_difference = 2(offset)$$

$$(T_2 - T_1) - (T_4 - T_3) = T_2 - T_1 - T_4 + T_3 = 2(offset)$$

$$offset = (T_2 - T_1 - T_4 + T_3) \div 2$$

PTP: Example



delay = 40
offset = 235
... but we don't know this yet

$T_2 - T_1$ = 1100-825 = 275 = *delay + offset*

$T_4 - T_3$ = 925-1120 = -195 = *delay – offset*

275 – (-195) = 470 = *2(offset)*

offset = 470/2 = 235

Time is set to 1225 - offset

when we receive last msg

= 1225 – 235 = **990**

master_slave_difference = $T_2 - T_1$ = *delay + offset*

slave_master_difference = $T_4 - T_3$ = *delay – offset*

master_slave_difference – slave_master_difference = 2(offset)

offset = $(T_2 - T_1 - T_4 + T_3) \div 2$

NTP vs. PTP

- Range
  - NTP: nodes widely spread out on the Internet
  - PTP: local area networks
    - Usually implemented at the physical layer to eliminate OS & scheduling overhead
- Accuracy
  - NTP usually several milliseconds on WAN
  - PTP usually sub-microsecond on LAN (around 1 μs)
    - PTP can be 10,000x more precise than NTP!

The End