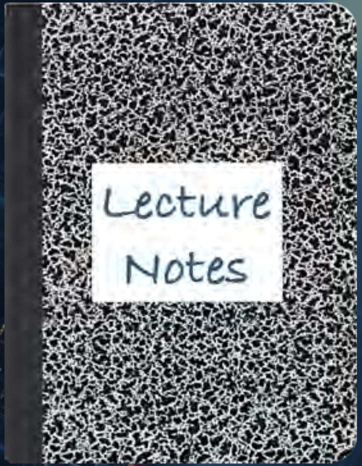
CS 417 – DISTRIBUTED SYSTEMS



Week 3: Synchronization

Part 1: Clock Synchronization

Paul Krzyzanowski

Synchronization

Synchronization covers coordinating interactions among distributed processes

Clocks	Identify *when* something happened
Logical clocks	Identify the sequence of events
Mutual exclusion	Only one entity can do an operation at a time
Leader election	Who coordinates activity? Who takes over?
Consistency/Agreement	Does everyone have the same view of events?

All of these are easy in non-distributed systems

All of these have challenges in distributed systems

Why do we care?

Distributed systems don't share a clock – each computer has its own

Clock Synchronization:

- Enable process to identify "**now**" consistent with other processes on other systems … and the real world

Wall time refers to the actual, real-world time.

Why do we care?

- Logging messages
- Checking deadlines and cache expirations
- Applications where time-based billing or access control is needed
- Checking expiration on certificates, authentication tokens, web cookies

UTC: Temps Universel Coordonné

The world's time standard

UTC(k): Collection of Atomic Clocks: (national labs)

Continuous time scale

TAI
International Atomic Time

Average of 450+ clocks

UTC
Universal Coordinated Time

Earth Rotation Observations (national labs)

UT1

Continuous time International Earth Rotation and Reference Systems Service

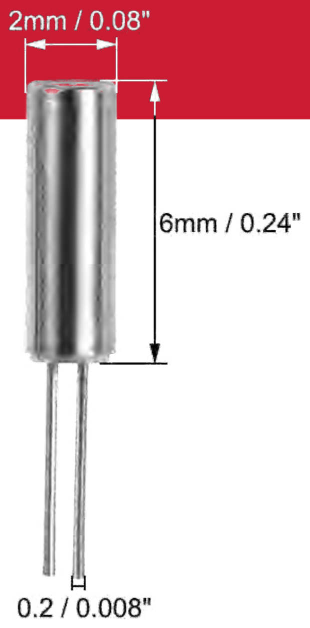
Not uniform

Leap Second
$|UT1 - UTC| < 0.9s$

Leap seconds are announced by the International Earth Rotation and Reference Systems Service (IERS)

How Computers Keep Time

- **Battery-Backed Real-Time Clock**
  - Quartz oscillator, typically 32,768 Hz
  - Keeps time across power-off
  - Used to initialize the system clock at boot



https://amzn.to/4kqUmF3

- **System Clock**
  - The operating system maintains a software system clock
  - High-resolution counter (e.g., TSC on x86, ARM Generic Timer)
  - OS converts counter ticks to seconds/nanoseconds
  - CPU driven by a fixed crystal clock (e.g., 38.4 MHz for a Core Ultra Series 3)

The Epoch

Epoch: fixed reference point

- Timestamps count elapsed time since epoch
- **Unix epoch:** 1970-01-01 00:00:00 UTC
- **Windows epoch:** 1601-01-01 00:00:00 UTC
  - Used by Windows FILETIME and NTFS, 100 ns intervals
- Why use an epoch?
  - Avoids time zone and DST ambiguity
  - Easy arithmetic and ordering: sorting, adding time, comparisons

Clock Imperfections

- Quartz time imperfections
  - Quartz oscillator frequency varies with tolerances, temperature, aging, environment
  - This makes time deviate from “true time”
- Typical PC quartz: about 50 ppm → about 4.32 s/day

Without sync, two machines drifting oppositely can differ by almost 9 seconds after one day

ppm = parts per million
50 ppm=50×10^{-6}
Seconds per day: 24×60×60=86,400 s
Daily drift: $86{,}400 \times 50 \times 10^{-6}$ = 4.32 s/day

Offset & Drift

- **Offset**: current difference from reference:
  - `offset = our_time - utc_time`
- **Drift**: rate error
  - clock runs slightly fast/slow, often expressed in ppm

Offset is the how off our time is right now
Drift is why the offset grows after a sync

Compensation

- **Synchronize**
  - Contact a server to find what the time *should* be
  - Now you have to set it
- **Adjust**
  - **Slew**: Gradually adjust clock rate without time going backward
  - **Step**: Jump clock for large offsets – but can break software assumptions
  - Apply **ongoing adjustment** to the clock frequency to limit drift
- **Repeat**:
  - Do this periodically to keep the offset minimal

See the Linux *adjtimex* system call

Accuracy, Precision, and Resolution

- **Accuracy**
  - Closeness to true UTC (absolute error)
- **Precision**
  - Consistency (low jitter)
  - A clock can be precise but offset from UTC
- **Resolution**
  - Smallest representable increment
  - High resolution does not imply accuracy/precision

Note that the NTP spec uses "precision" to refer to resolution.

Synchronization Algorithms

Why Not Attach a GNSS Receiver to Each System?

- **Not practical for most systems**
  – Antenna needs a view of the sky
  – Receivers need to wait for a fix
  – Accuracy gets worse near buildings, bridges, trees, …
  – Deployment cost scales poorly (installation, cabling, antenna placement)
  – Another dependency that can fail and can be attacked
  – Power hungry: Android & iOS use NTP, even with a GPS

GNSS = Global Navigation Satellite System
{GPS, GLONASS, Galileo, and BeiDou}

- **Chip-scale atomic clock**
  – Nice, but around $2,000+
  – Most computers won't have this either
  – And even if you have it, you still need to set it to give you the right time

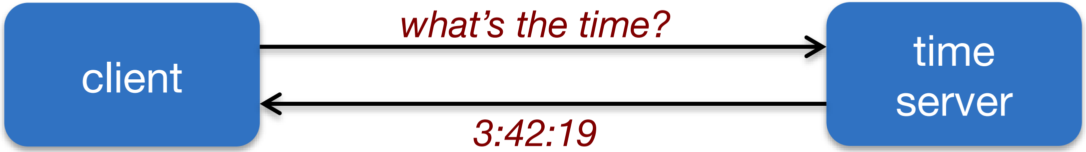
A Simple Request-Response Approach

Simplest synchronization technique

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

`ping time.google.com`

10.8 – 29.9 ms response
Average ping time = 17.52 ms

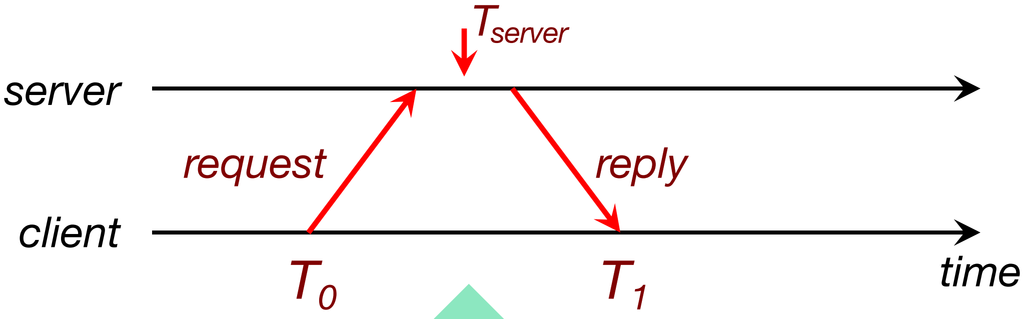


Does not account for network or processing latency

Cristian's Algorithm

Compensate for delays

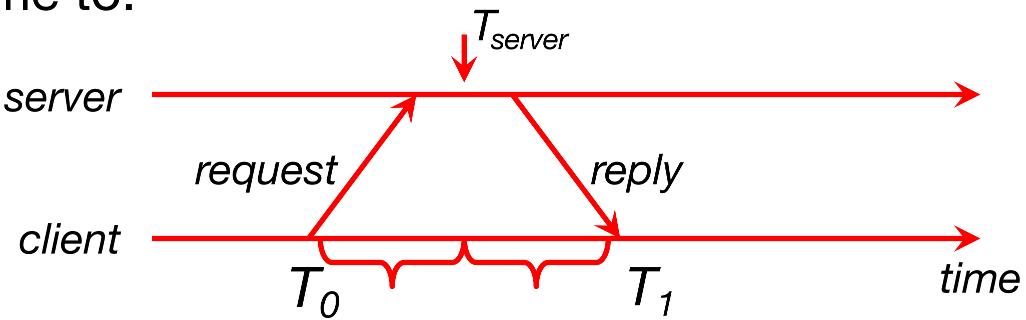
- Request sent: T_0
- Reply received: T_1
- Timestamp from server: T_{server}
- Assume network delays are symmetric



Assume the server creates T_{server} in the middle of the time interval

Cristian's Algorithm

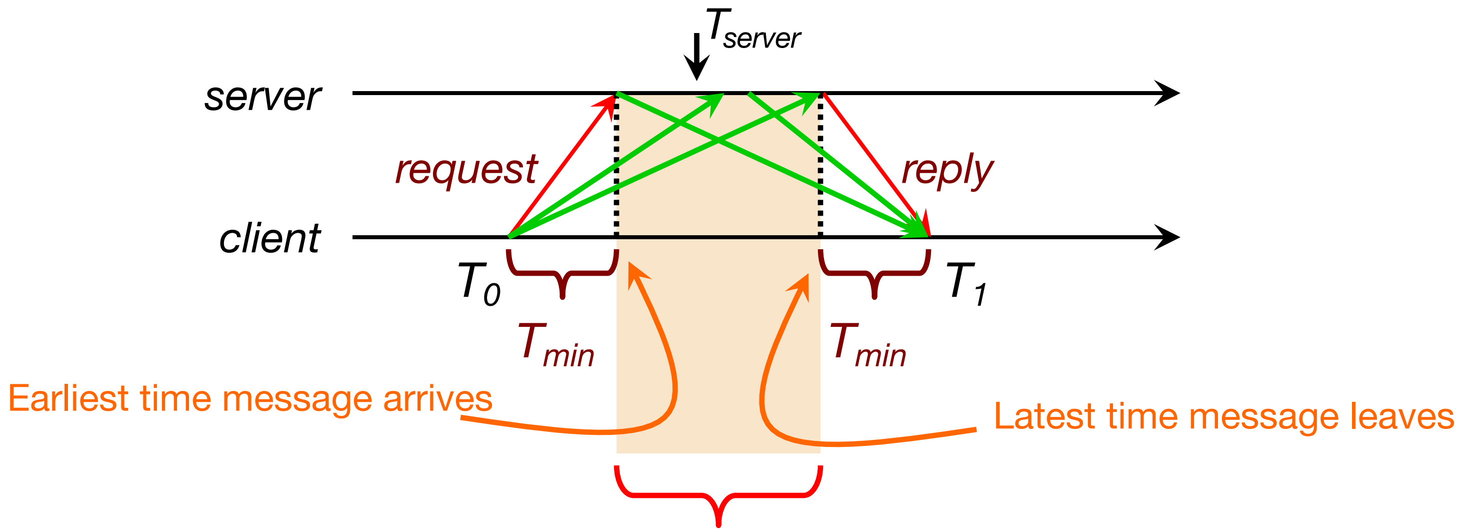
Client sets time to:



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

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

Error bounds



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

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_{server}

- Total elapsed time is 800ms
- At LEAST 200ms was used by the network in each direction
- At LEAST 400ms will always be used in the network
- We have 800-400, or 400ms that we're not sure about
  - Since the timestamp is set to the middle, that's ±200ms uncertainty

$T_0 = 5{:}08{:}15.100$

$T_1 = 5{:}08{:}15.900$

$T_s = 5{:}09{:}25{:}300$

$T_{min} = 200$ ms

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

Note: errors are additive

Network Time Protocol, NTP

Enable clients across Internet to be accurately synchronized to UTC despite message delays

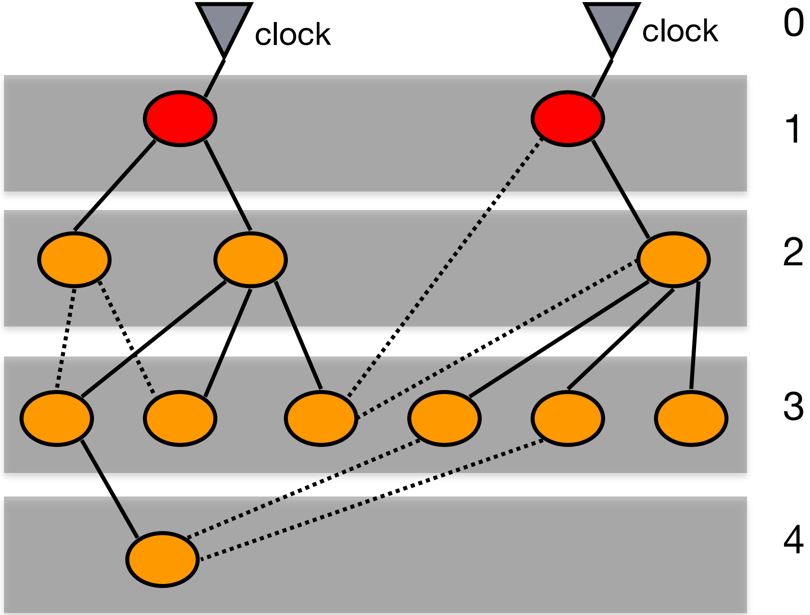
- Use statistical techniques to filter data and gauge quality of results

- 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

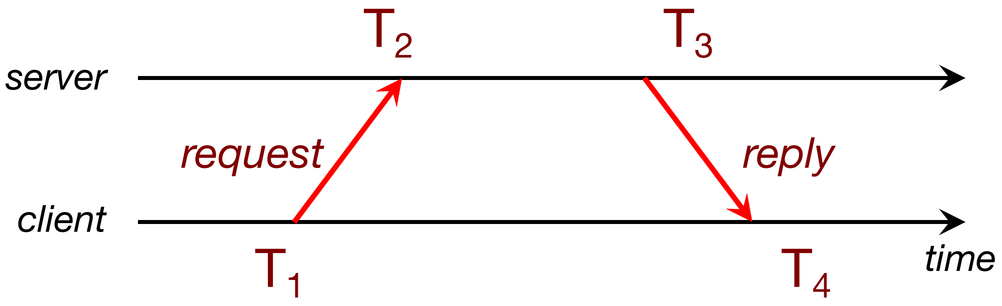
Arranged in ***strata***

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



Synchronization Subnet

NTP Messages



Round-trip network delay:

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

Time offset:

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

Collect many (θ, δ) pairs – prefer low delay and low jitter

NTP: Getting and Setting the Time

- **Query** multiple servers
  - Reject outliers (faulty or bad time)
- Favor sources with **lower jitter and dispersion**
  - Create a weighted average of the remaining offsets
- **Discipline** local clock
  - Slew for small offsets (typically < 128 ms)
  - Slew for large offsets (typically > 128 ms)

UDP, not TCP!

Why?
- TCP delays transmission
- Processing overhead
- Retransmissions destroy symmetric latency!

Precision Time Protocol (PTP)

More accurate clock synchronization

Sometimes NTP isn't good enough

- 5G networks (phase sync)
- Industrial process control: synchronizing actuators/sensors
- Financial trading timestamps
- Power-grid synchrophasors (voltage, frequency, current, phase angle)
- Audio/video sync

- NTP issues
  - NTP timestamps are captured after OS/network delays – vary with load

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 generated at the network card to minimize delay and jitter
    - Reduces jitter to nanosecond scale

- 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)

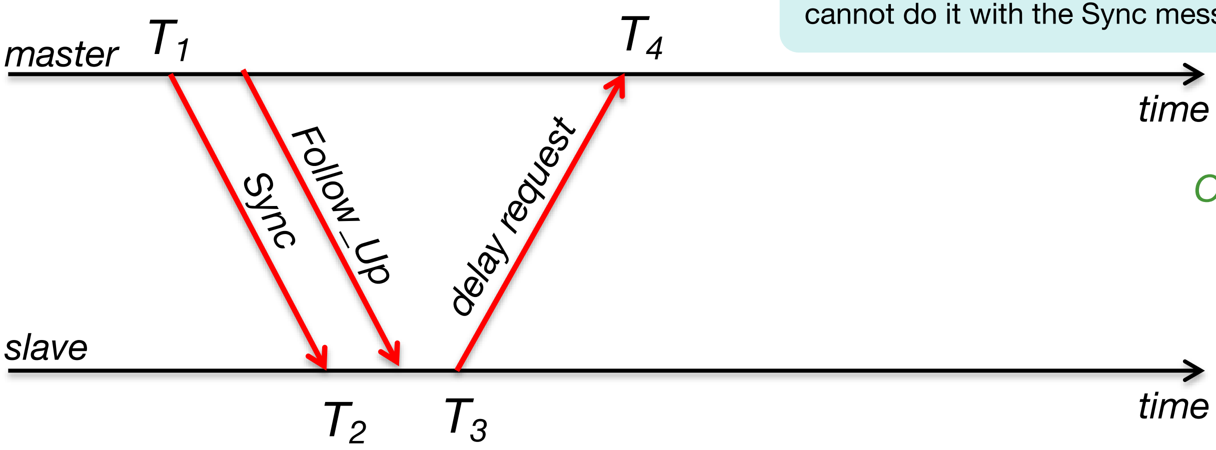
- Two phases in synchronization
  1. Offset correction
  2. Delay correction

PTP: Choose the “best” clock - **BMCA**

Best Master Clock Algorithm

- 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: Send delay request



Follow_Up sends T_1 because some hardware cannot do it with the Sync message

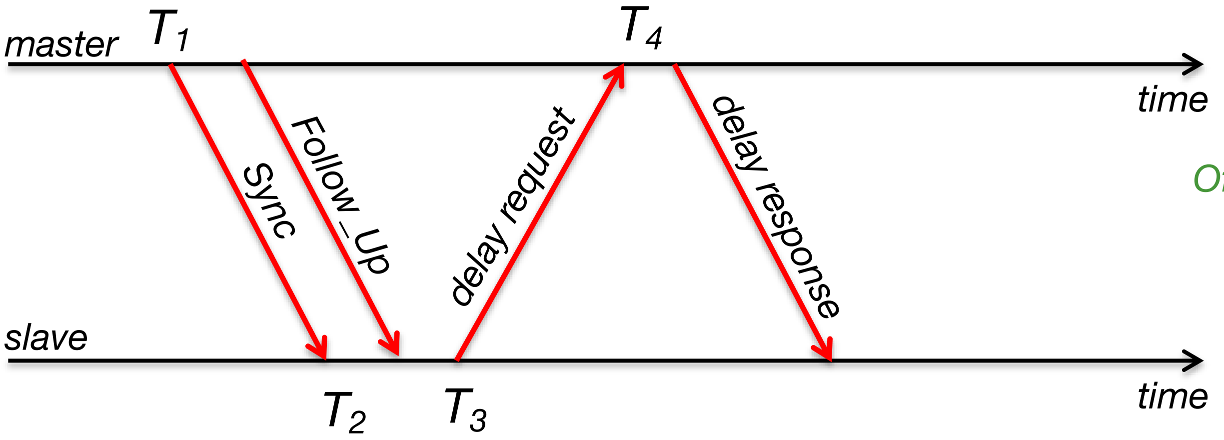
$Offset + Delay = T_2 - T_1$

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

Note the time it was sent

PTP assumes network delays are symmetric!

PTP: Receive delay response

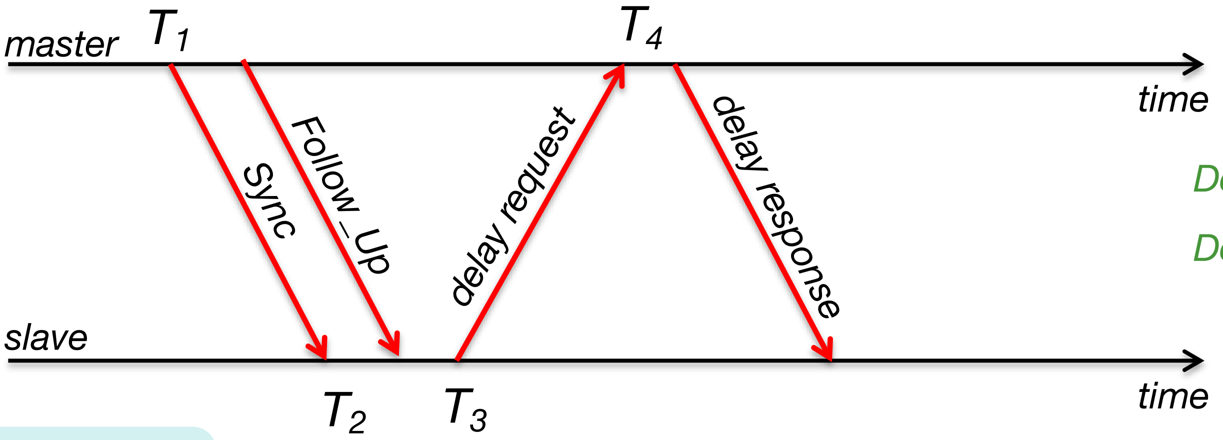


Offset + Delay = T_2 - T_1

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

Delay response = Delay - Offset = $T_4 - T_3$

PTP: Slave computes offset



Delay + Offset = T_2 - T_1

Delay - Offset = T_4 - T_3

The messages give us 2 equations with 2 unknowns: delay & offset

$$\textit{master\_slave\_difference} = T_2 - T_1 = \textit{delay} + \textit{offset}$$

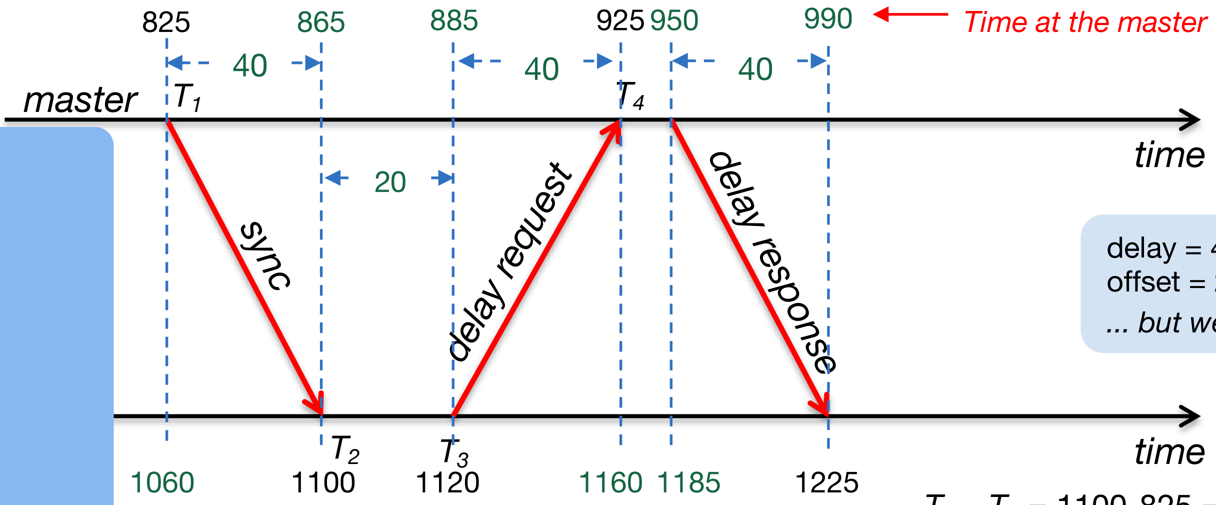
$$- \quad \textit{slave\_master\_difference} = T_4 - T_3 = \textit{delay} - \textit{offset}$$

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

$$(T_2 - T_1) - (T_4 - T_3) = T_2 - T_1 - T_4 + T_3 = 2(\textit{offset})$$

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

PTP: Example



$T_1 = 825$
$T_2 = 1100$
$T_3 = 1120$
$T_4 = 925$

Offset = 235
Set time to
T_4-offset = 990

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

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

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

$275 - (-195) = 470 = 2(offset)$

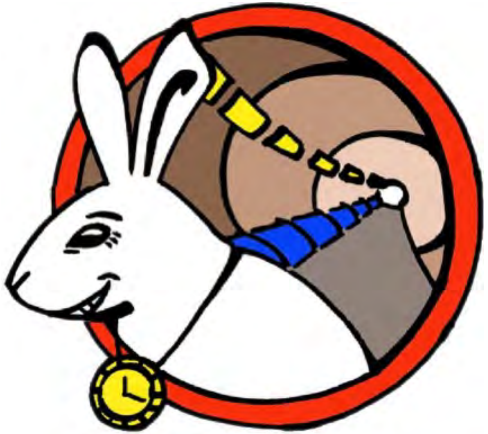
$offset = 470/2 = 235$

Time is set to 1225 - offset
= 1225 – 235 = **990**

when we receive last msg

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

White Rabbit



The Large Hadron Collider at CERN

- Timestamps data from thousands of detectors
- Needed higher precision than PTP

White Rabbit

- Extension to PTP
- Uses Synchronous Ethernet for ultra-low, predictable latency
- Sub-nanosecond accuracy

The End