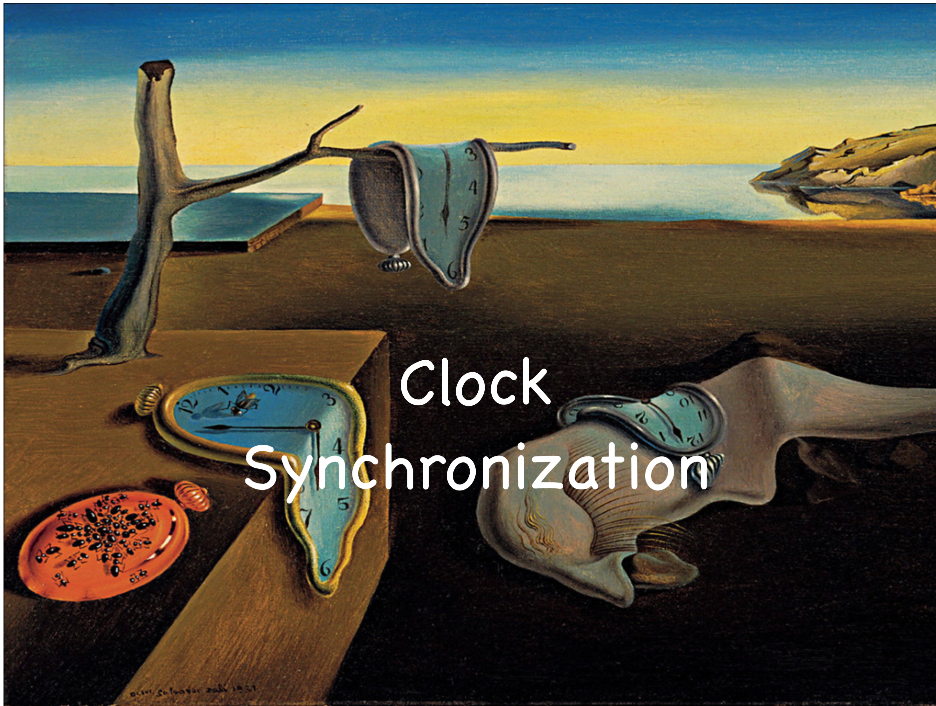


Byzantine generals v2.0

Attack at midnight!

Chaaaaaarge!

ZZZZZ

0:00

23:30

Clock drift

1

2

3

4

5

6

Clock drift

- Bound on drift: ρ

$$(1-\rho)(t-t') \leq H(t) - H(t') \leq (1+\rho)(t-t')$$

- ρ is typically small (10^{-6})
  - $\rho^2 \approx 0$
  - $\frac{1}{1-\rho} = 1 + \rho$
  - $\frac{1}{1+\rho} = 1 - \rho$

$H(t)$ (clock time)

$(1+\rho)t$

$(1-\rho)t$

t (real time)

External vs internal synchronization

External Clock Synchronization:

keeps clock within some maximum deviation from an **external time source**.

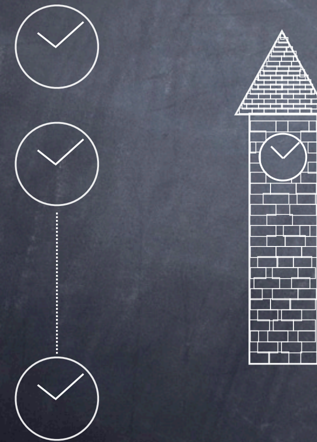
- exchange of info about timing events of different systems
- can take actions at **real-time** deadlines

Internal Clock Synchronization:

keeps clocks within some maximum deviation from **each other**.

- can measure **duration** of distributed activities that start on one process and terminate on another
- can totally order events that occur in a distributed system

Probabilistic Clock Synchronization (Cristian)

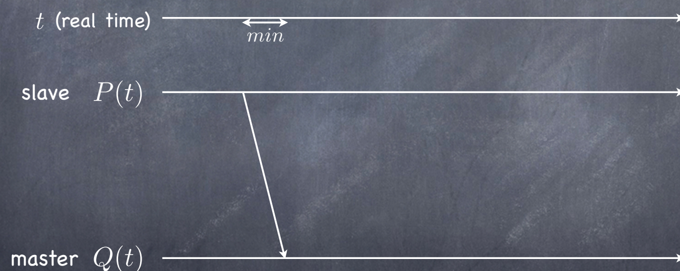


- Master-Slave architecture
- Master can be connected to external time source
- Slaves read master's clock and adjust their own

How accurately can a slave read the master's clock?

Setup and assumptions

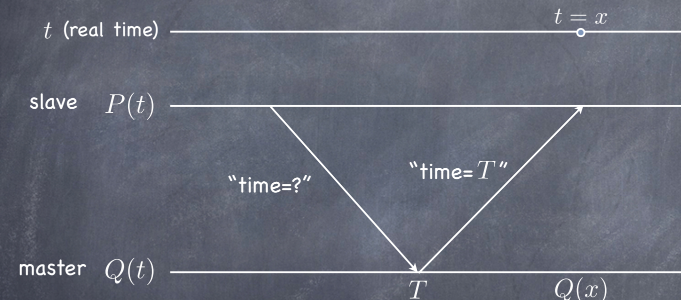
Goal: Synchronize the slave's clock with the master



Assume that minimum delay is known

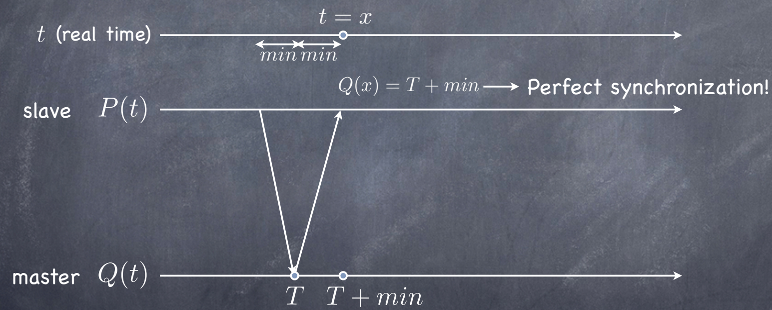
Assume that clock drifts are known (ρ for both)

The protocol



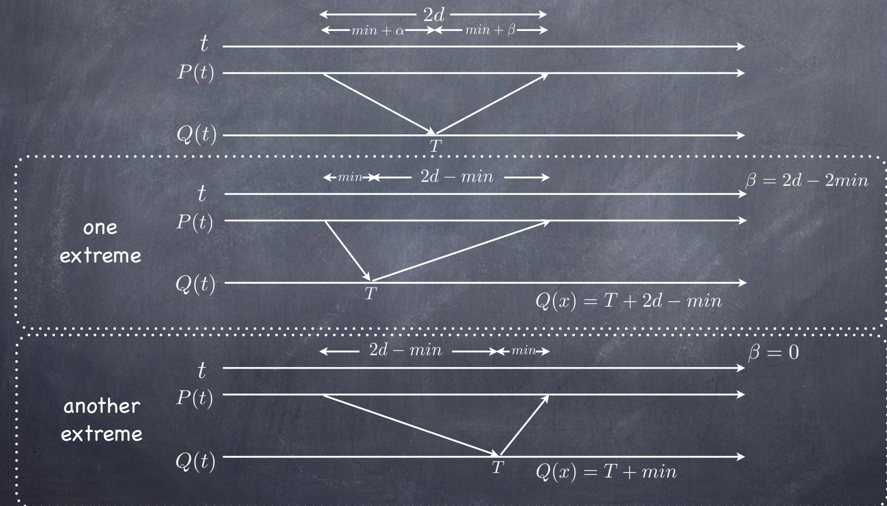
Question: what is $Q(x)$?

Ideal scenario

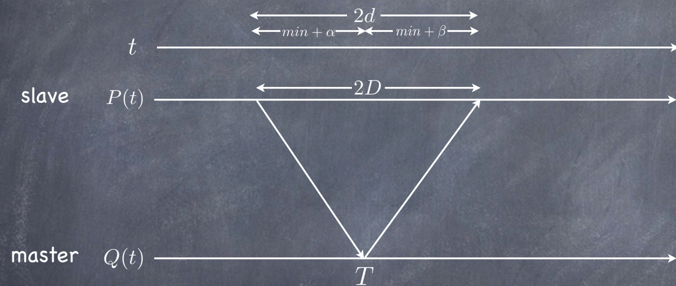


Assume no clock drift

Problem #1: message delay

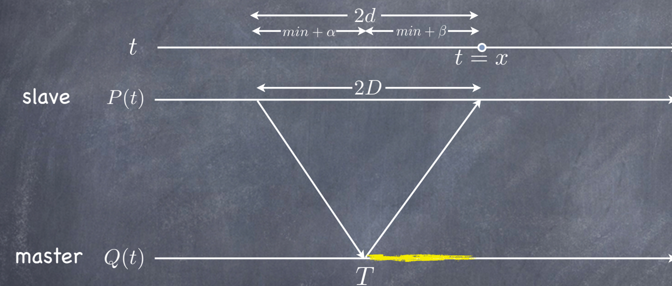


Problem #2: slave drift



$$2d(1-\rho) \leq 2D \leq 2d(1+\rho)$$

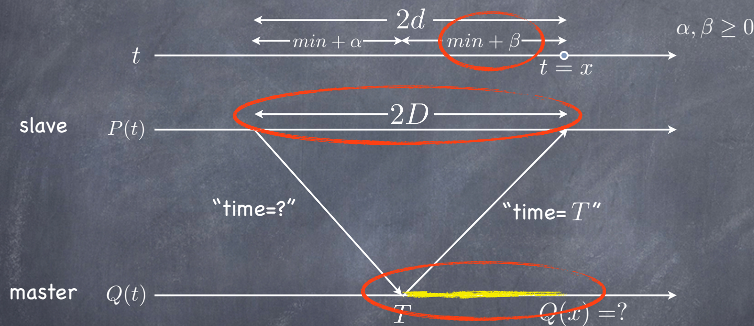
Problem #3: master drift



During ▬ the master's clock drifts

Even if you know β, there is still some uncertainty!

Cristian's algorithm



Cristian's algorithm

Naive estimation: $Q(x) = T + (min + \beta)$

(take master's drift into account)

$Q(x) \in [T + (min + \beta)(1-\rho), T + (min + \beta)(1+\rho)]$

$0 \leq \beta \leq 2d - 2min$ (take delay into account)

$Q(x) \in [T + (min+0)(1-\rho), T + (min+2d-2min)(1+\rho)]$
$= [T + (min)(1-\rho), T + (2d - min)(1+\rho)]$

$2d \leq 2D(1+\rho)$ (take slave's drift into account)

$Q(x) \in [T + (min)(1-\rho), T + (2D(1+\rho) - min)(1+\rho)]$
$= [T + (min)(1-\rho), T + 2D(1+2\rho) - min(1+\rho)]$

Slave's estimation and precision

Slave's best guess: $Q(x) = T + D(1+2\rho) - min \cdot \rho$

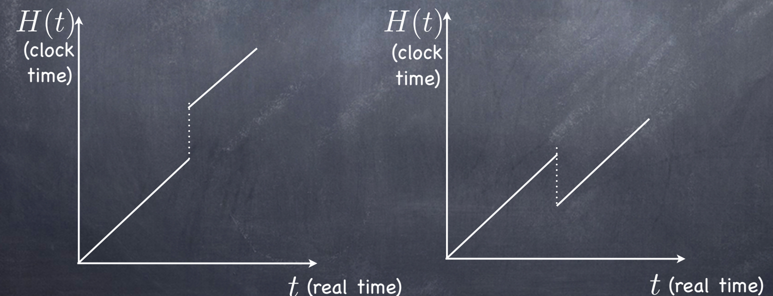
Maximum error: $e = D(1+2\rho) - min$

You can keep trying, until you achieve the required precision

Adjusting the clock

After synchronizing:

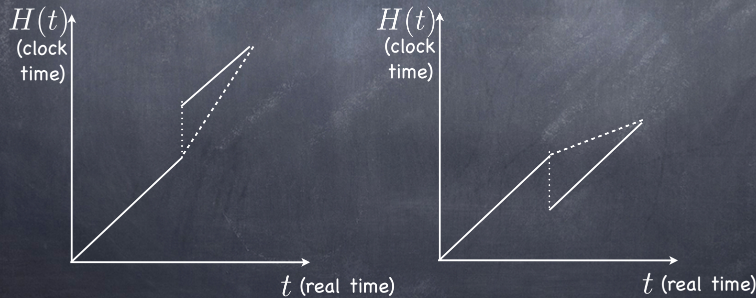
- If slave simply sets $P(x) = Q(x)$, it could create time discontinuities.



Adjusting the clock

Logical clock $C(t) = H(t) + A(t)$

Hardware clock (points to $H(t)$) — Adjustment function (points to $A(t)$)



Network Time Protocol

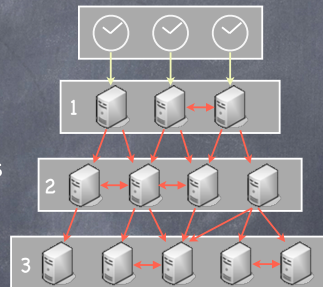
- The oldest distributed protocol still running on the Internet
- Hierarchical architecture
- Latency-tolerant, jitter-tolerant, fault-tolerant.. very tolerant!

Hierarchical structure

Each level is called a "stratum"

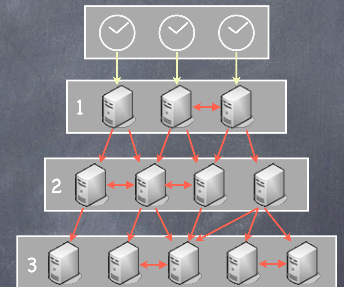
- Stratum 0: atomic clocks
- Stratum 1: time servers with direct connections to stratum 0
- Stratum 2: Use stratum 1 as time sources and work as servers to stratum 3
- etc....

Accuracy is loosely coupled with stratum level



Very tolerant. How?

- Tolerance to jitter, latency, faults: **redundancy**
- Each machine sends NTP requests to many other servers on the same or the previous stratum
- The synchronization protocol between two machines is similar to Cristian's algorithm
- Each response defines an interval $[T_1,T_2]$
- How to **combine** those intervals?



Marzullo's algorithm

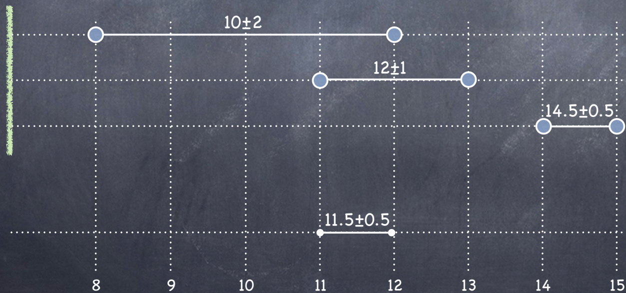
- Given M source intervals, find the largest interval that is contained in the largest number of source intervals

Marzullo's algorithm

- Given M source intervals, find the largest interval that is contained in the largest number of source intervals

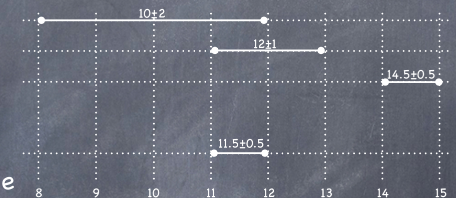
The intuition

- Visit the endpoints left-to-right
- Count how many source intervals are active at each time
  - Increase count at starting points, decrease at ending points



Preprocessing

- For each source interval [T_1,T_2], create 2 tuples of the form <time, type>:
  - <T_1,+1> (start of interval)
  - <T_2,-1> (end of interval)
- Sort all tuples according to time



Example:
Source intervals: [8,12], [11,13], [14,15]
Tuples: <8,+1> <12,-1> <11,+1> <13,-1> <14, +1> <15, -1>
Sorted: <8,+1> <11,+1> <12,-1> <13,-1> <14, +1> <15, -1>

The algorithm

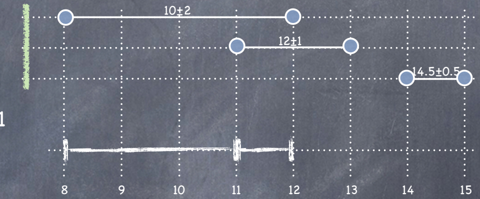
```
count=0, max=0
for all tuples<time[i],type[i]> {
        count = count + type[i]

        if(count>max) {
            max=count
            winStart=time[i]
            winEnd=time[i+1]
        }
}
return [winStart, winEnd]
```

Notes:

- *count*: numbers of "active" intervals
- *max*: maximum numbers of "active" intervals we have seen
- *count=count+type[i]* : if it's a startpoint (type=+1), increase count, else decrease it
- *if(count>max)* : if this is the highest number of active intervals we have seen, let the winning interval be (*time[i], time[i+1]*)
  - If the next point is a starting point, it will **replace** the current winning interval
  - If the next point is an ending point, it will **end** this winning interval

The algorithm at work

Sorted: <8,+1> <11,+1> <12,-1> <13,-1> <14, +1> <15, -1>



Init: count=0, max=0

<8,+1> : count = count + (+1) = 1
Is count>max? Yes
max=1, winStart=8, winEnd=11

<11,+1> : count = count + (+1) = 2
Is count>max? Yes
max=2, winStart=11, winEnd=12

<12,-1> : count = count + (-1) = 1
Is count>max? No

<13,-1> : count = count + (-1) = 0
Is count>max? No

<14, +1> : count = count + (+1) = 1
Is count>max? No

<15, -1 : count = count + (-1) = 0
Is count>max? No

return [11,12]

NTP timestamps

How to represent time?
"Friday October 25th 2013, 16:15:00" ?
"20131025161500CDT" ?

NTP: 64-bit UTC timestamp

32 bits	32 bits
offset in seconds	sub-second precision

offset = #seconds since January 1, 1900

Wraps around every 2^{32} seconds = 136 years
First wrap-around: 2036
Solution: 128-bit timestamp. "Enough to provide unambiguous time representation until the universe goes dim"