Time and Ordering I

ECE 599 / CS 519 – SPRING 2015

Why Synchronization?

- **You want to catch a bus at 6.05 pm, but your watch is off by 15 minutes**
  - What if your watch is Late by 15 minutes?
    - You'll miss the bus!
  - What if your watch is Fast by 15 minutes?
    - You'll end up unfairly waiting for a longer time than you intended

- **Time synchronization is required for both**
  - Correctness
  - Fairness

Synchronization In The Cloud

- Cloud airline reservation system
- Server A receives a client request to purchase last ticket on flight ABC 123.
- Server A timestamps purchase using local clock 9h:15m:32.45s, and logs it. Replies ok to client.
- That was the last seat. Server A sends message to Server B saying "flight full."
- B enters "Flight ABC 123 full" + its own local clock value (which reads 9h:10m:10.11s) into its log.
- Server C queries A's and B's logs. Is confused that a client purchased a ticket at A after the flight became full at B.
- This may lead to further incorrect actions by C

Why is it Challenging?

- End hosts in Internet-based systems (like clouds)
  - Each have their own clocks
  - Unlike processors (CPUs) within one server or workstation which share a system clock
- Processes in Internet-based systems follow an *asynchronous* system model
  - No bounds on
    - Message delays
    - Processing delays
  - Unlike multi-processor (or parallel) systems which follow a *synchronous* system model

Some Definitions

- An Asynchronous Distributed System consists of a number of processes.
- Each process has a state (values of variables).
- Each process takes actions to change its state, which may be an instruction or a communication action (send, receive).
- An event is the occurrence of an action.
- Each process has a local clock – events *within* a process can be assigned timestamps, and thus ordered linearly.
- But – in a distributed system, we also need to know the time order of events *across* different processes.

Clock Skew vs. Clock Drift

- **Each process (running at some end host) has its own clock.**
- **When comparing two clocks at two processes**:
  - Clock Skew = Relative Difference in clock *values* of two processes
    - Like distance between two vehicles on a road
  - Clock Drift = Relative Difference in clock *frequencies (rates)* of two processes
    - Like difference in speeds of two vehicles on the road
- **A non-zero clock skew implies clocks are not synchronized.**
- **A non-zero clock drift causes skew to increase (eventually).**
  - If faster vehicle is ahead, it will drift away
  - If faster vehicle is behind, it will catch up and then drift away

How often to Synchronize?

- Maximum Drift Rate (MDR) of a clock
- Absolute MDR is defined relative to Coordinated Universal Time (UTC). UTC is the "correct" time at any point of time.
  - MDR of a process depends on the environment.
- Max drift rate between two clocks with similar MDR is 2 * MDR
- Given a maximum acceptable skew M between any pair of clocks, need to synchronize at least once every: M / (2 * MDR) time units
  - Since time = distance/speed

External vs. Internal Synchronization

Consider a group of processes

- **External Synchronization**
  - Each process C(i)'s clock is within a bound D of a well-known clock S external to the group
  - $|C(i) - S| < D$ at all times
  - External clock may be connected to UTC (Universal Coordinated Time) or an atomic clock
  - E.g., Cristian's algorithm, NTP
- **Internal Synchronization**
  - Every pair of processes in group have clocks within bound D
  - $|C(i) - C(j)| < D$ at all times and for all processes i, j
  - E.g., Berkeley algorithm

External vs. Internal Synchronization

- **External Synchronization with D => Internal Synchronization with 2*D**

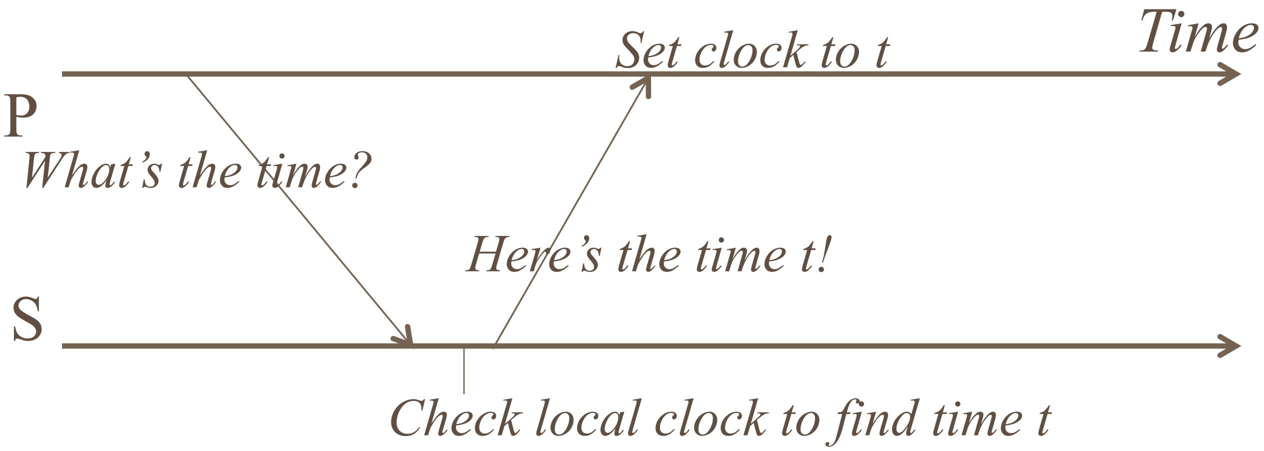
- **Internal Synchronization does not imply External Synchronization**
- **In fact, the entire system may drift away from the external clock S!**

Next

- Algorithms for Clock Synchronization
- Cristian's Algorithm

Basics

- **External time synchronization**
- **All processes P synchronize with a time server S**

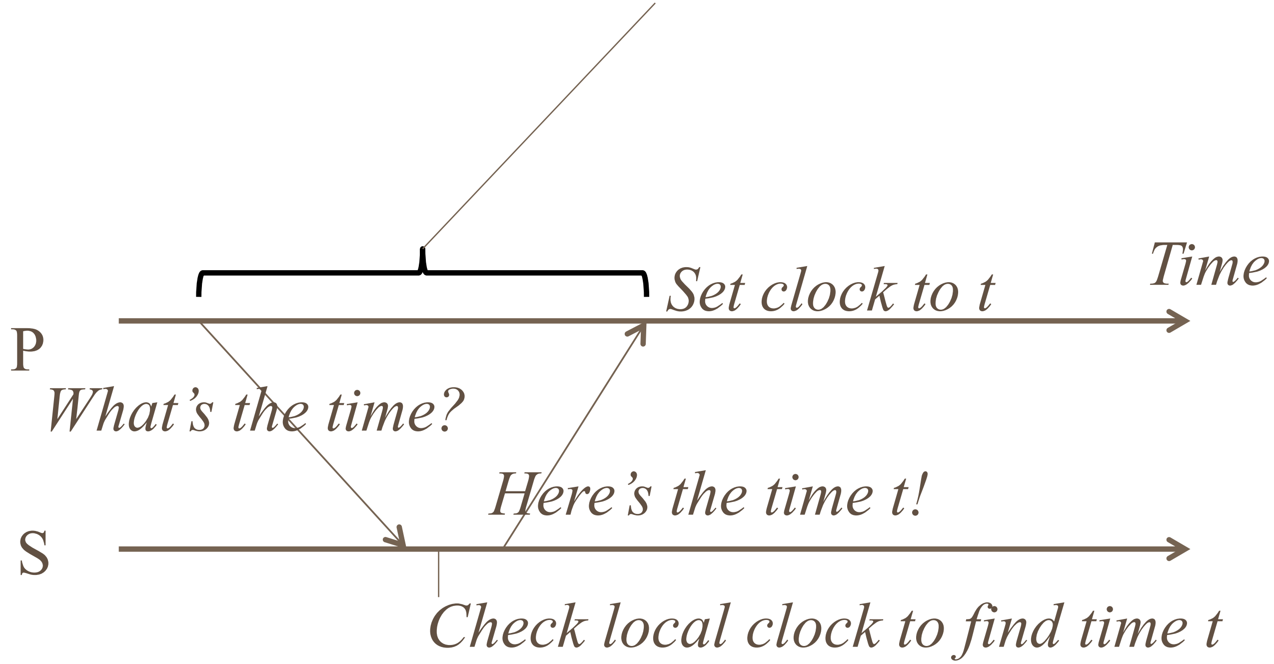


What's Wrong

- By the time response message is received at P, time has moved on
- P's time set to *t* is inaccurate!
- Inaccuracy a function of message latencies
- Since latencies unbounded in an asynchronous system, the inaccuracy cannot be bounded

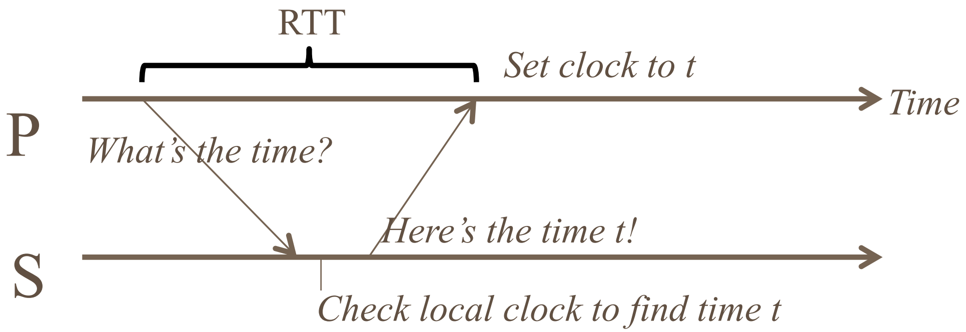
Cristian's Algorithm

- P measures the round-trip-time RTT of message exchange



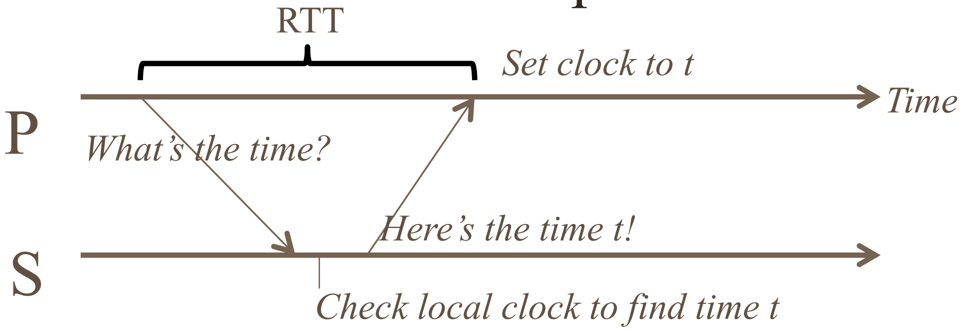
Cristian's Algorithm (2)

- **P measures the round-trip-time RTT of message exchange**
- **Suppose we know the minimum P → S latency min1**
- **And the minimum S → P latency min2**
  - min1 and min2 depend on Operating system overhead to buffer messages, TCP time to queue messages, etc.



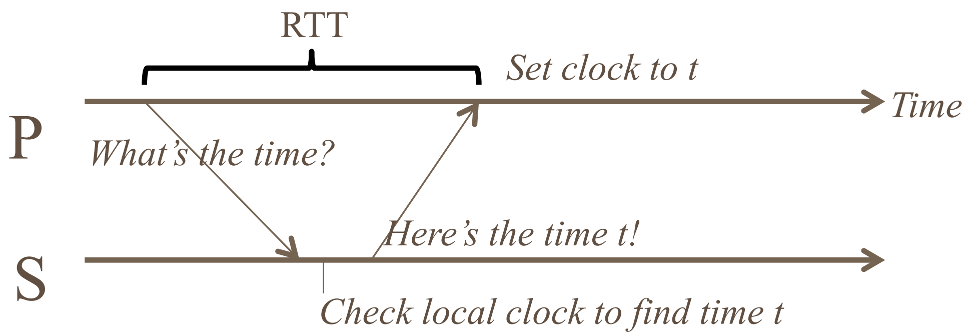
Cristian's Algorithm (3)

- P measures the round-trip-time RTT of message exchange
- Suppose we know the minimum P → S latency min1
- And the minimum S → P latency min2
  - min1 and min2 depend on Operating system overhead to buffer messages, TCP time to queue messages, etc.
- The actual time at P when it receives response is between [t+min2, t +RTT-min1]



Cristian's Algorithm (4)

- The actual time at P when it receives response is between [t+min2, t+RTT-min1]
- P sets its time to halfway through this interval
  - To: t + (RTT+min2-min1)/2
- Error is at most (RTT-min2-min1)/2
  - Bounded!



Gotchas

- **Allowed to increase clock value but should never decrease clock value**
  - May violate ordering of events within the same process

- **Allowed to increase or decrease speed of clock**

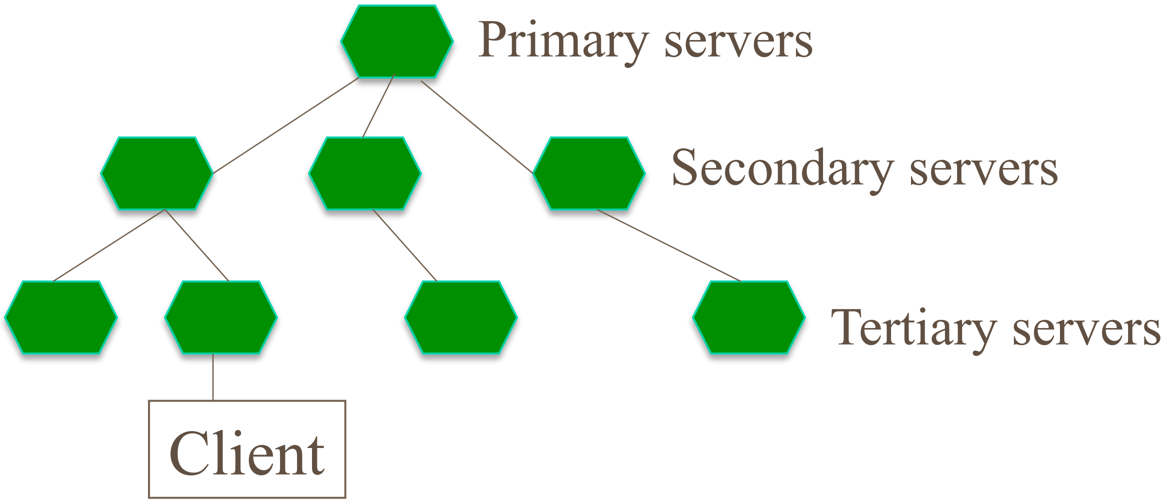
- **If error is too high, take multiple readings and average them**

Next

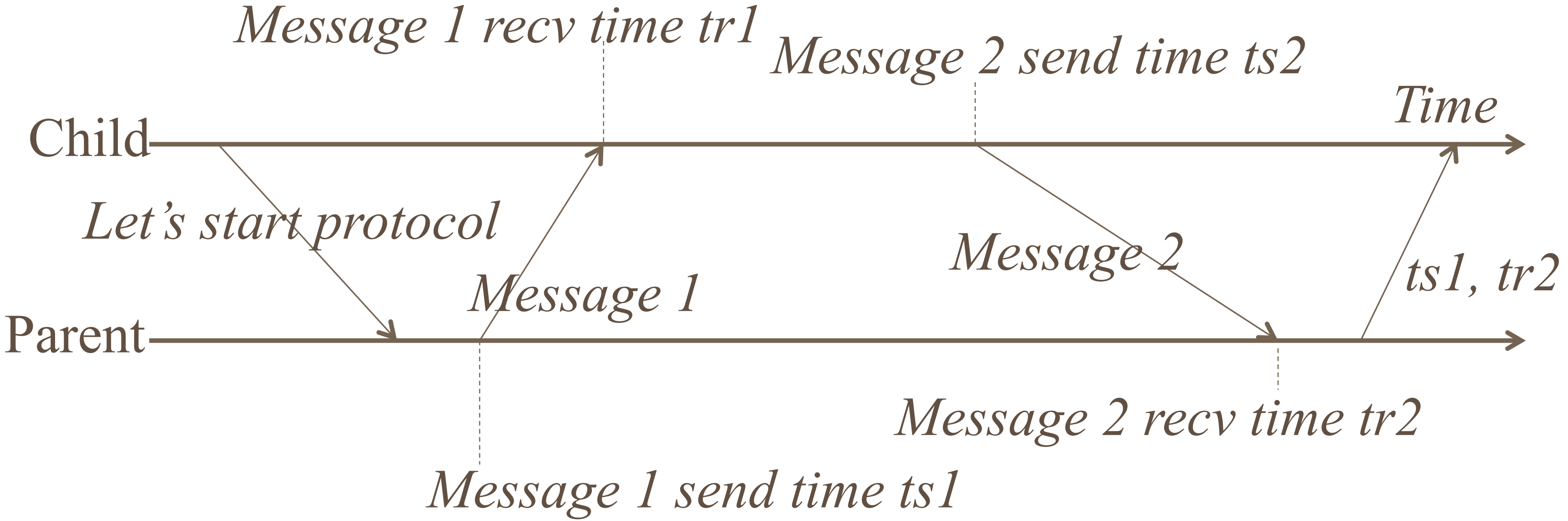
- Algorithms for Clock Synchronization
- NTP

NTP = Network Time Protocol

- NTP Servers organized in a tree
- Each Client = a leaf of tree
- Each node synchronizes with its tree parent



NTP Protocol



What the Child Does

- Child calculates *offset* between its clock and parent's clock
- Uses $ts1$, $tr1$, $ts2$, $tr2$
- Offset is calculated as

  $$o = (tr1 - tr2 + ts2 - ts1)/2$$

Why o = (tr1 - tr2 + ts2 - ts1)/2?

- Offset $o = (tr1 - tr2 + ts2 - ts1)/2$
- Suppose real offset is *oreal*
  - Child is ahead of parent by *oreal*
  - Parent is ahead of child by -*oreal*
- Suppose one-way latency of Message 1 is *L1* (*L2* for Message 2)
- No one knows *L1* or *L2*!
- Then
  - $\boldsymbol{tr1 = ts1 + L1 + oreal}$
  - $\boldsymbol{tr2 = ts2 + L2 - oreal}$

Why o = (tr1 - tr2 + ts2 - ts1)/2? (2)

- Then

$$tr1 = ts1 + L1 + oreal$$

$$tr2 = ts2 + L2 - oreal$$

- Subtracting second equation from the first

$$oreal = (tr1 - tr2 + ts2 - ts1)/2 + (L2 - L1)/2$$

$$\Rightarrow oreal = o + (L2 - L1)/2$$

$$\Rightarrow |oreal - o| < |(L2 - L1)/2| < |(L2 + L1)/2|$$

  - Thus, the error is bounded by the round-trip-time

And yet...

- **We still have a non-zero error!**
- **We just can’t seem to get rid of error**
  - Can’t, as long as message latencies are non-zero
- **Can we avoid synchronizing clocks altogether, and still be able to order events?**

Time and Ordering I: Summary

- **Clocks are unsynchronized in an asynchronous distributed system**
- **But need to order events, across processes!**
- **Time synchronization**
  - Cristian's algorithm
  - NTP
  - Berkeley algorithm
  - But error a function of round-trip-time

Next

- Time and Ordering: Logical Clocks