Time

Network Distributed Computing

Mark Cummins, Institute of Technology Blanchardstown

Objectives

- To give the student some intuition and mathematical tools for thinking about the execution of distributed systems, by exploring the notions of physical and logical time and global states.

- To give students an appreciation of the function of synchronised clocks in distributed systems, and of the problem of delays in networks that can prevent accurate synchronisation.

Objectives

- To understand the key features of Cristian's synchronisation algorithm, the Berkeley algorithm and the Network Time Protocol.

- To understand the utility of logical clocks (Lamport and vector), the rules for updating them and their limitations.

Introduction

- Time is an important practical issue.
  - For example:
    - We often want to measure time accurately to know the *order* events take place.

- Some algorithms depend on *clock synchronisation*,
  - for example
    - use of timestamps in transactions
    - checking authenticity of requests
    - eliminating duplicate updates

Introduction

- Time is also an important theoretical construct but is problematic in distributed systems.
- *Each computer may have its own physical clock and clocks typically deviate and we can not synchronise them perfectly.*
- We shall look at algorithms for synchronising clocks approximately and then look at logical clocks and vector clocks, which are a tool for ordering events without knowing precisely when they occurred.

Introduction

- The absence of global physical time makes it difficult to find out the state of distributed programs as they occur.

- We often need to know what state process A is in when process B is in a certain state, but we cannot rely on physical clocks to know what is true at the same time.

Clocks

- A computer contains its own physical clock based on an oscillating crystal.

- The operating system reads the hardware clock value $\mathbf{H_i(t)}$, scaling it and adding an offset, so as to produce a software clock $\mathbf{C_i(t)}$ that approximately measures real physical time **t** for process $\mathbf{p_i}$.

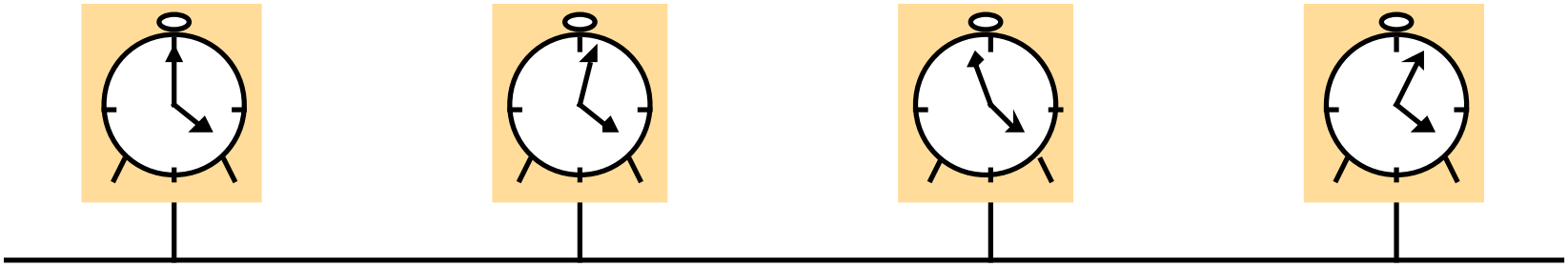
- In other words, if the real time in an absolute frame of reference is t, then the reading on the software clock is $\mathbf{C_i(t)}$.

Clocks

- For example, $\mathbf{C_i(t)}$ could be the 64-bit value of the number of nanoseconds that have elapsed at time **t** since a convenient reference time.
- In general, the clock is not completely accurate so $C_i(t)$ will differ from **t**.
- However, if $\mathbf{C_i}$ behaves sufficiently well, we can use its value to timestamp any event at p_i.
- Note also that successive events will correspond to different timestamps only if the ***clock resolution*** (period between updates of the clock value) is smaller than the ***time interval*** between successive events.

Clock Skew and Clock Drift

- The instantaneous difference between the readings of any number of clocks is called their ***skew***.



- ***Clock drift*** means the clocks count time at different rates.

Clock Skew and Clock Drift

- The clock's ***drift rate*** is the change in the offset (difference in reading) between the clock and a nominal perfect reference clock per unit of time measured by the reference clock, e.g. ~10^{-6} seconds/second

- or 1 sec every 11.6 days for ordinary clocks and 10^{-7} 0r 10^{-8} for high precession quartz clocks.

Coordinated Universal Time (UTC)

- Computer clocks can be synchronised to external sources of highly accurate time.
- Most accurate time keepers use the atomic oscillators with drift rate of approximately 10^{-13} sec $\rightarrow$ International Atomic Time (1967).
- The standard second is defined as 9,192,631,770 periods of transition between the two hyperfine levels of the ground state of Caesium 133 atom.
- Normal everyday time is based on Astronomical Time which varies due to astronomical effects and gets out of step with IAT.

Coordinated Universal Time (UTC)

- UTC is an international standard for time-keeping, based on atomic time but adjusted with leap seconds to keep in step with astronomical time.

- UTC signals are synchronised and broadcast regularly from land-based radio stations (0.1 - 10 ms accuracy) and satellites (GPS 1 ms accuracy) all over the world.

- Computers using receivers can synchronise time with GPS or alternatively use a National Institute for Standards time (talking clock) via the telephone lines which is accurate to a few milliseconds.

Synchronising Physical Clocks

- In order to know what time of day events occur in a Distributed System, it is necessary to synchronise the processes' clocks with an authoritative external source of time.

Synchronising Physical Clocks

External synchronisation (uses external source of time)

- For a synchronisation bound D > 0 and for a source S of UTC time, $|S(t) - C_i(t)| < D$ for i=1,2, ... , N and for all real-times t in I (interval of real-time).

- If the clocks are synchronised with one another to a known degree of accuracy then we can measure the interval between two events occurring on different computers by appealing to their local clocks.

Synchronising Physical Clocks

Internal synchronisation (synchronise clocks with one another)

- For a synchronisation bound D > 0,
  - $|C_i(t) - C_j(t)| < D$
  - for i,j=1,2, ... , N and for all real-times t in I.

- In other words, clocks C_i agree within the bound D.

Synchronising Physical Clocks

- Clocks that are internally synchronised are not necessarily externally synchronised.

- Hardware clocks H are though to be correct if its drift rate falls within a known bound, $\rho > 0$,
  - e.g. (10^{-6} seconds per second).

Synchronising Physical Clocks

- A clock that does not keep to a correctness condition are ***faulty***.

- A clock's ***crash failure*** is said to occur when the clock stops ticking altogether,

- any other clock failure is an ***arbitrary failure***, e.g. Y2K bug.

Algorithms for Internal and External Synchronisation

Synchronisation in a synchronous system

- Looking at the simplest case where we have internal synchronisation between two processes in a synchronous distributed system. We known the following:
  - bounds for drift rate
  - maximum time for message transfer delay
  - time to execute each step of a process

Algorithms for Internal and External Synchronisation

- To synchronise clocks, one process sends the time t on its local clock to the other process in a message m. The receiving process then sets its clock to time t + T where *T is the time taken to transmit m between two processes.*

- Unfortunately T is subject to variation and is unknown.
  - Let U = uncertainty in message transmission time (= max - min)
  - max = upper bound on transmission time
  - min = lower bound on transmission time

- If **T = (max-min)/2** then the skew is **(max-min)/2** or **U/2**

Algorithms for Internal and External Synchronisation

- In general, for ***synchronous systems***, the optimum bound that can be achieved on clock skew when synchronising N clocks is **U(1-(1/N))** where N>1.

- Most distributed systems used in practice are ***asynchronous***; the factors leading to message delays are not bounded in their effect and there is *no upper bound maximum on message transmission delays*.

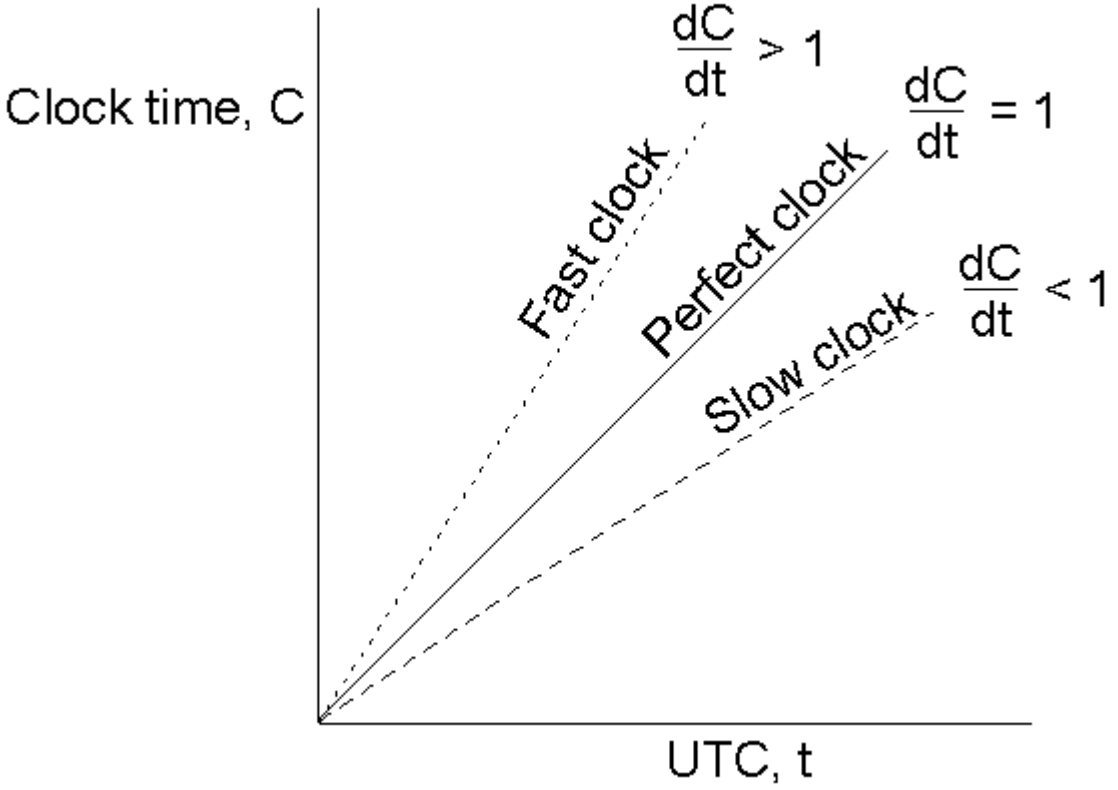
Algorithms for Internal and External Synchronisation

- For asynchronous systems (like the Internet),
  - **T = min + x**
  - **where x >=0**
- The value of x is not known but a distribution of values may be measurable for a particular installation.

Time

- Time is complicated in a distributed system.
- Physical clocks run at slightly different rates so they can 'drift' apart.
- Clock makers specify a max. drift rate ρ
- By definition: $1-\rho \leq {}^{dC}/_{dt} \leq 1+\rho$
  where C(t) is the clock's time as a function of the real time.

Clock Synchronization



The relation between clock time and UTC when clocks tick at different rates.

23

Clock Synchronisation

- $1-\rho \leq dC/dt \leq 1+\rho$
- A perfect clock has $dC/dt = 1$
- Assuming two clocks have the same max drift rate ρ. To keep them synchronized to within a time interval delta, δ, they must re-sync every $\delta/2\rho$ seconds.

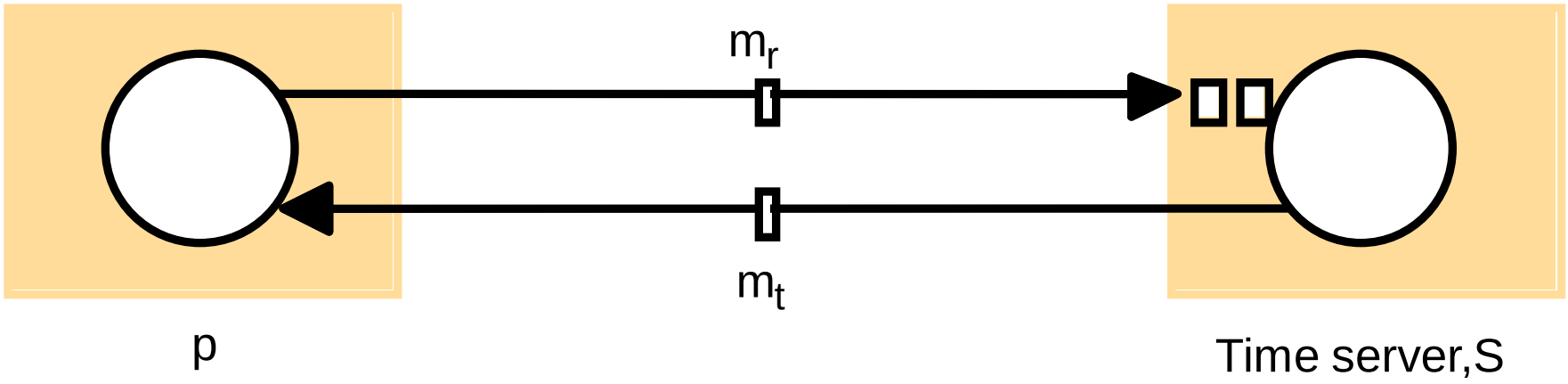
Cristian's method for Synchronising Clocks

- Use a time server with UTC time to synchronise computers externally.

- Upon request, the server process S supplies the time according to its clock.

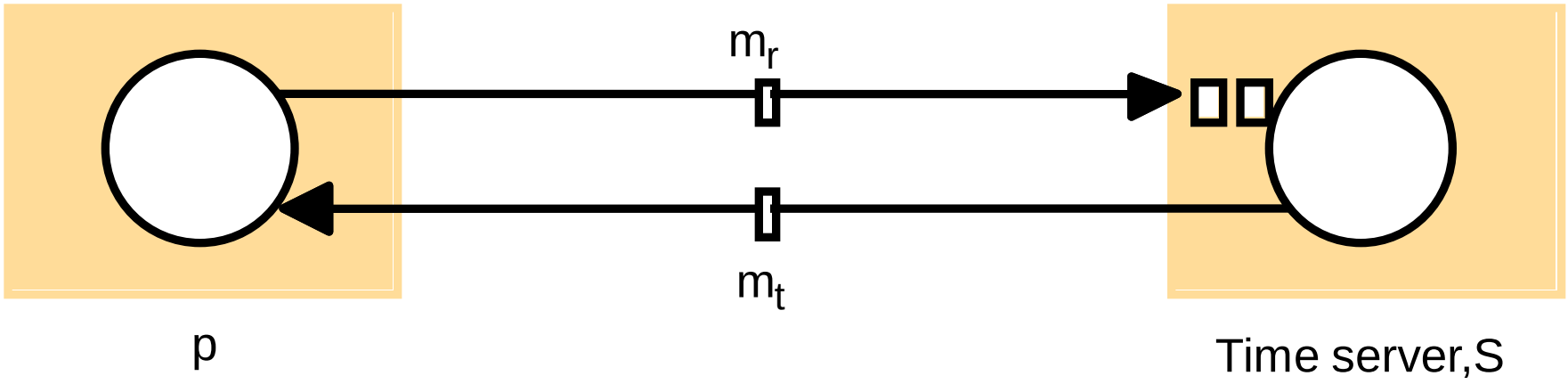
Cristian's method for Synchronising Clocks

- Cristian observed that round-trip times for message exchange between pairs of processes is often reasonably short (fraction of a second) compared to the required accuracy - Probabilistic algorithms.



Cristian's method for Synchronising Clocks

- A process p requests the time in a message m_r and receives the time value t in the message m_t. Process p records the total round-trip time T_{round} taken to send the request and receive the reply.



Cristian's method for Synchronising Clocks

- Sample values for T_{round} are 1 to 10 milliseconds on a LAN over which time a clock with a drift rate of 10^{-6} seconds/second varies by at most 10^{-5} milliseconds.

- A simple estimate of the time to which p should set its clock is

$$\mathbf{(t + T_{round}/2).}$$

Cristian's method for Synchronising Clocks

- If two messages are transmitted over different networks then *if the minimum transmission time is known* or can be conservatively estimated, then we have the following situation:
  - The earliest point at which S could have placed the time in m_t was "min" after p dispatched m_r.
  - The latest point at which it could have done this was "min" before m_t arrived at p.
  - The time the clock on S when the reply message arrives is therefore in the range

$$[t+min, t+T_{round}-min]$$

Cristian's method for Synchronising Clocks

- The width of this range is T_{round}-2*min (i.e. subtract them).
- So the accuracy is ± (T_{round}/2)-min
- Variability can be dealt with to some extent by making several requests to S and taking the minimum value of T_{round} to give the most accurate estimate.
- The greater is the accuracy required, the smaller is the probability of achieving it. This is because the most accurate results are those in which both messages are transmitted in a time close to min - an unlikely event in a busy network.

Exercises

1. A client attempts to synchronise with a time server. It records the round-trip times and timestamps returned by the server in the table below. Which of these times should it use to set its clock? To what time should it set it? Estimate the accuracy of the setting with respect to the server's clock. If it is known that the time between sending and receiving a message in the system concerned is at least 8 ms, do your answers change?

Round-trip(ms)	Time (hr:min:sec)
22	10:54:23.674
25	10:54:25.450
20	10:54:28.342

2. If the system is required to synchronise a file server's clock to within ±1 ms, what is the required round trip time? What are the implications of having such a value for round trip times?

Discussion of Cristian's Algorithm

- Sample values for T_{round} are 1 to 10 milliseconds on a LAN over which time a clock with a drift rate of 10^{-6} seconds/second varies by at most 10^{-5} milliseconds.

- Depends on single server
  - failure means no synchronisation possible.

- Use group of time servers, each with own UTC time signal,
  - client could multicast request but only use first reply.

Discussion of Cristian's Algorithm

- Faulty server (or impostor) could wreak havoc in distributed system.
  - Cristian assumes external sources are self-checking.
- Dolev et al. [1986] showed that for a system to achieve agreement then
  - N > 3f where f is the number of faulty clocks out of a total of N.
- Malicious interference can be countered using authentication techniques.

Berkeley Algorithm

- Co-ordinator computer (master) periodically polls other computers (slaves) whose clocks are to be synchronised.
- Master collects the times of the slaves and estimates their local times by observing the round-trip times and averages the values obtained (including its own clock).
- The idea is that this average cancels out the individual clock's offset.
- The accuracy depends on the round-trip times.

Berkeley Algorithm

- Master sends the amount by which each individual slave's clock requires adjustment (can be positive or negative value).
- Faulty clocks are eliminated by ignoring clocks that differ from others by a significant amount.
- If master fails, another computer is elected to function as master.
- *Gusella and Zetti described an experiment using 15 computers whose clocks were synchronised to within about 20-25 msec using this protocol. The local drift rate was measured to be less than $2x10^{-5}$ and max round trip was 10 msec.*

The Network Time Protocol (NTP)

- Cristian's method and Berkeley algorithm are intended primarily for use with intranets.

- The Network Time Protocol defines an architecture for a time service (based on UTC) and a protocol to distribute time information over the Internet.

The Network Time Protocol (NTP)

NTP main design aims and features:

- To provide a service enabling clients across the Internet to be synchronised accurately to UTC. - (uses statistical methods for filtering time data and dealing with delays).
- To provide a reliable service that can survive lengthily losses of connectivity - uses redundant servers and paths.
- To enable clients to resynchronise sufficiently frequently to offset the rates of drift found in most computers - servers can scale to large no. of clients.
- To provide protection against interference with the time service, whether malicious or accidental - authentication techniques and validation of return addresses.

The Network Time Protocol (NTP)

- **NTP service provides a number of servers across the Internet in a hierarchical structure.**
  - At the top are *primary servers* connect directly to UTC source.
  - The next level has *secondary servers* are synchronised to primary servers.
  - The next level has computers synchronised to secondary servers and so on.
  - The bottom of the hierarchy is the *user's workstation*.
  - This structure allows for reconfiguration of computers in hierarchy if some become unreachable.

The Network Time Protocol (NTP)

- The use of the Network Time Protocol (NTP v3 - RFC 1305) has been proven by the Internet community to be a reliable method of synchronizing the time on a number of network connected hosts.

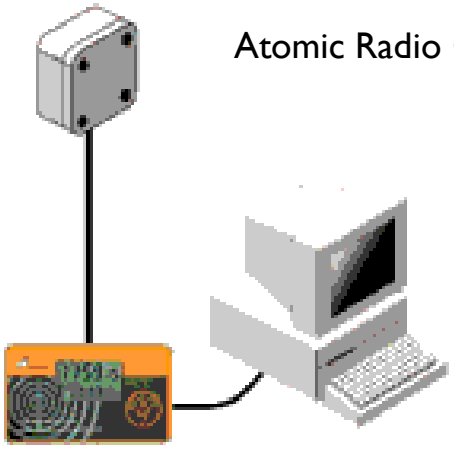
- Time synchronizing is required to allow meaningful comparison of performance and diagnostic data gathered from physically remote sites. In current practice, a small number of servers are provided with a high quality time reference, using either a short wave receiver tuned to the NIST Time reference signal, or a Global Positioning System (GPS) receiver.

The Network Time Protocol (NTP)

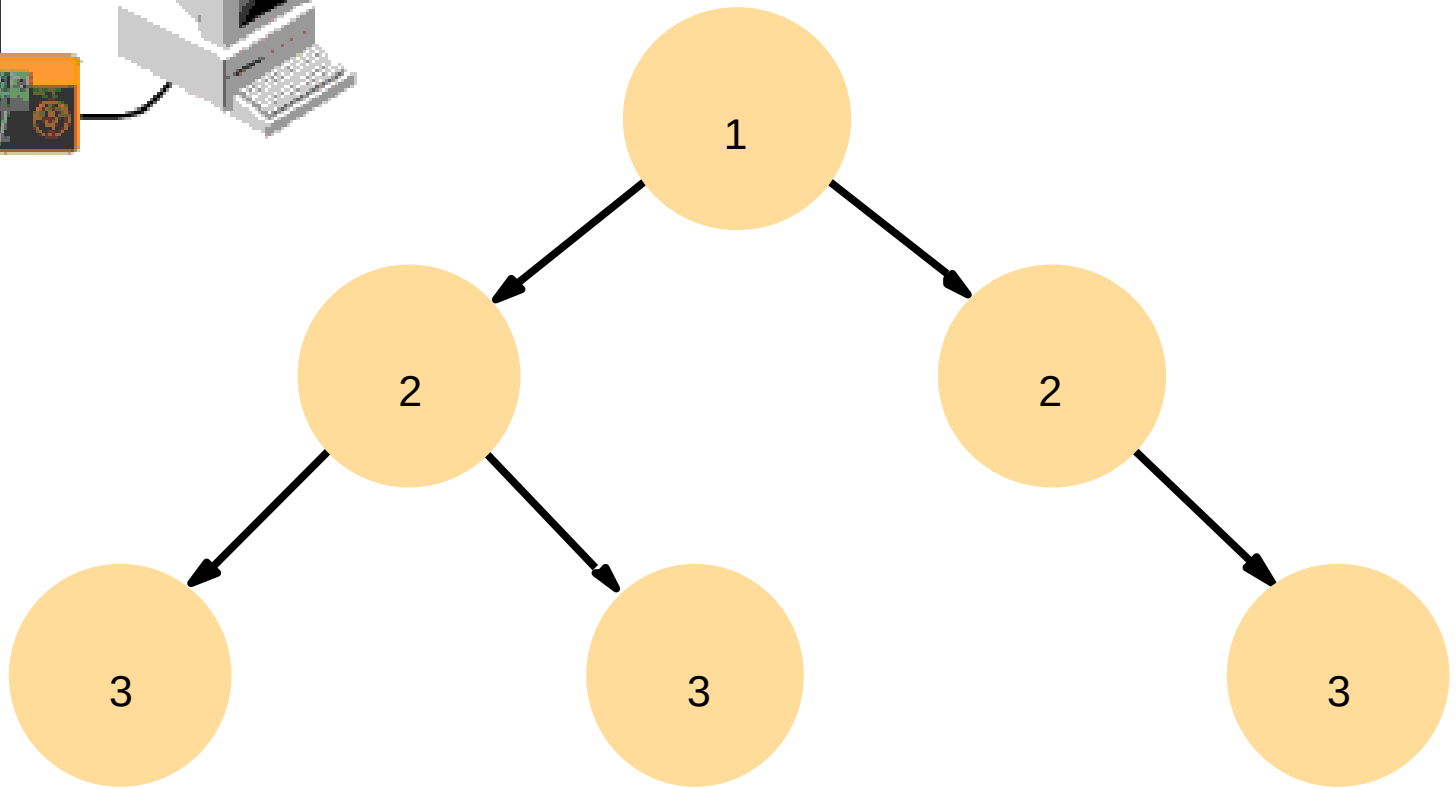
- Network attached hosts use NTP to derive the time from these reference servers. At the current time, the greatest factor in determining the quality of the synchronization is the topological distance from the NTP server to the host as determined by the number intermediate hops.

- Given the global nature of time, the use of Global Positioning System (GPS) for clocking provides the greatest flexibility for participants.

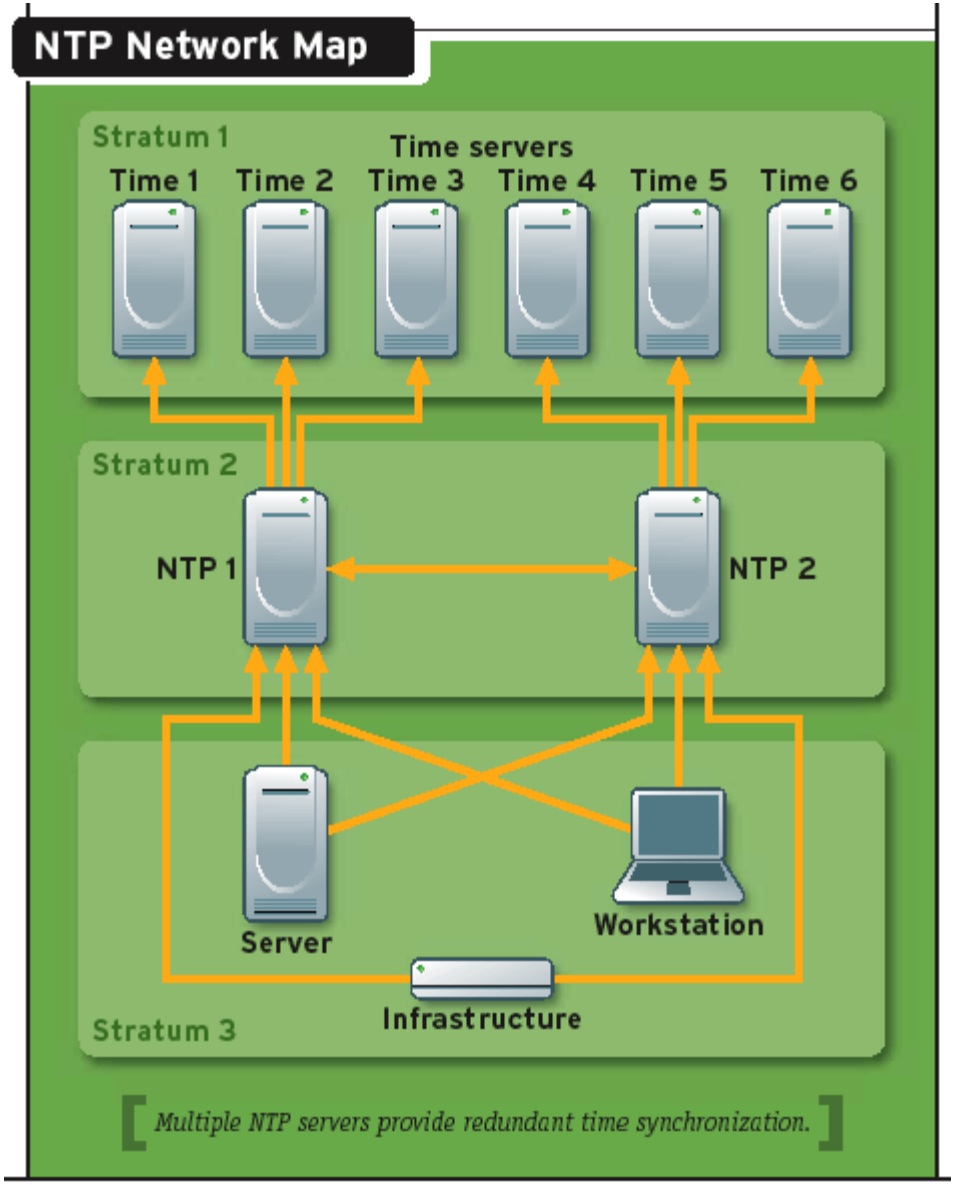
- The NTP timestamps will *overflow* in the year 2036.

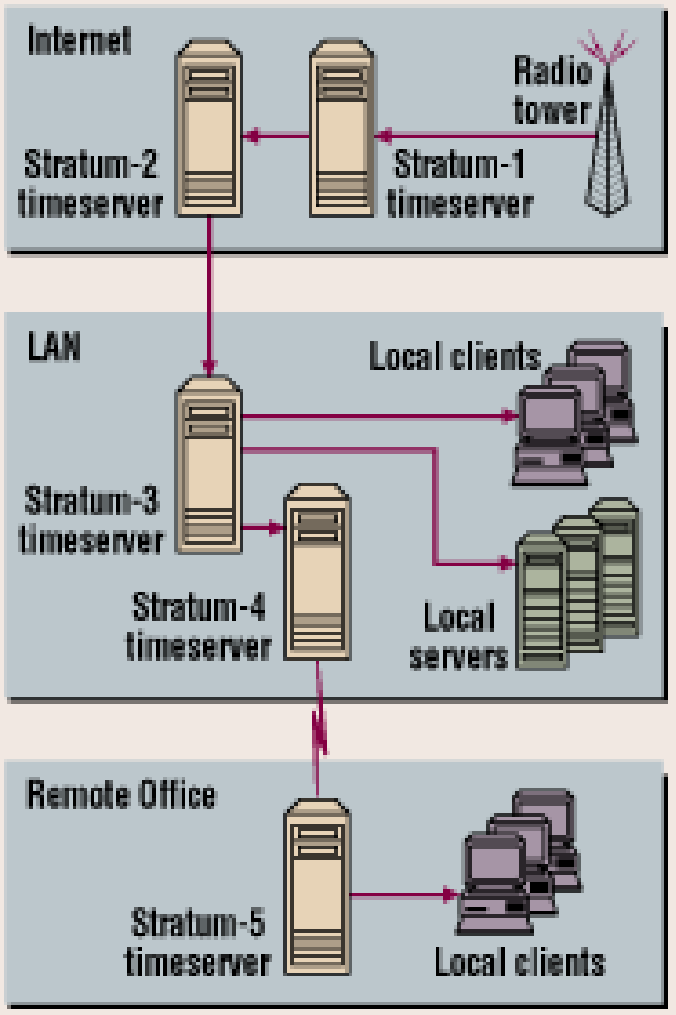


Atomic Radio Clock Receiver

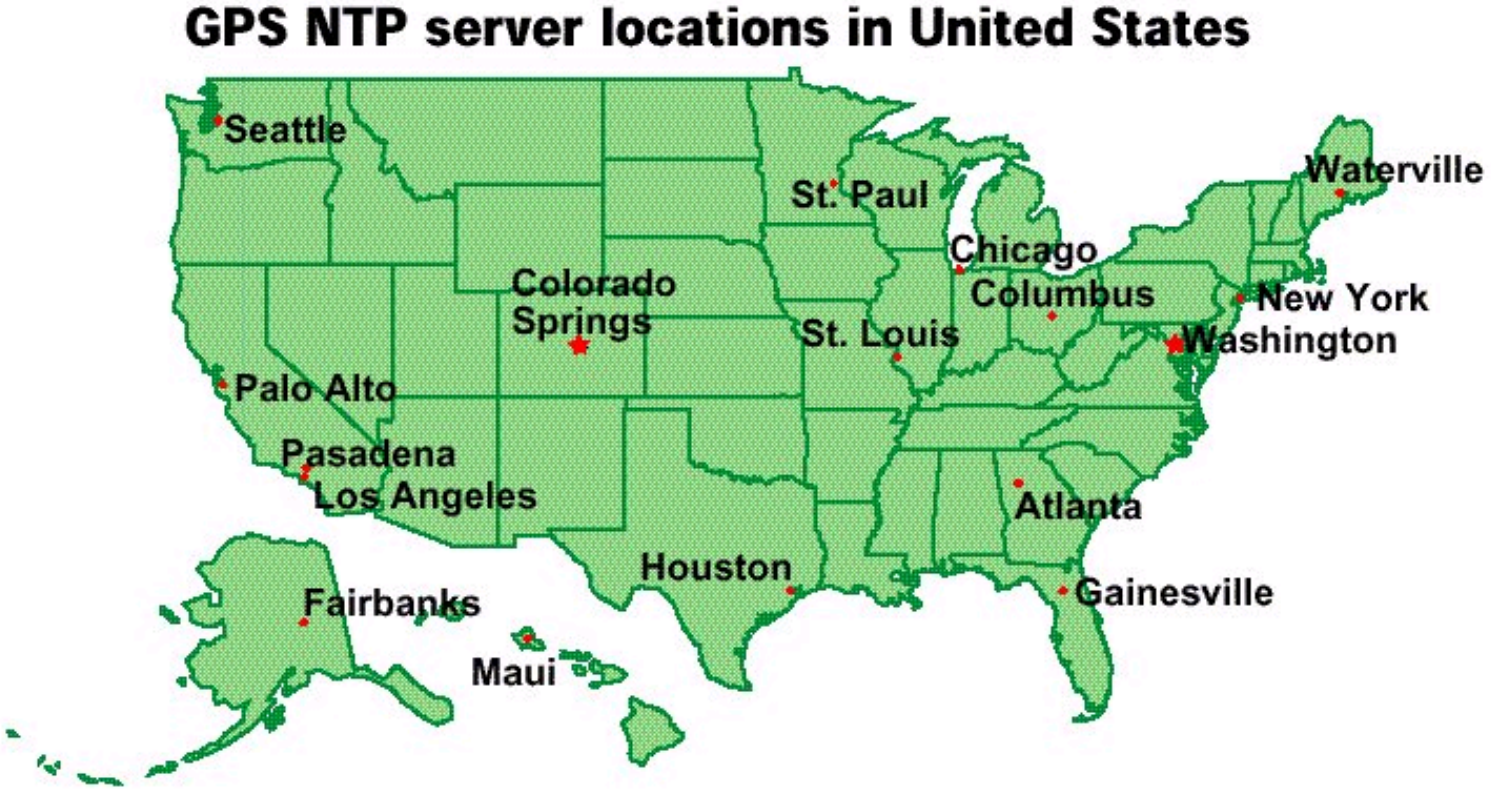


Note: Arrows denote synchronisation control, numbers denote strata.





Using multiple timeservers within the enterprise, administrators can effectively synchronize everything from routers to workstations.


GPS NTP server locations in United States
Seattle
St. Paul
Waterville
Chicago
Colorado Springs
Columbus
New York
St. Louis
Washington
Palo Alto
Pasadena
Los Angeles
Atlanta
Houston
Gainesville
Fairbanks
Maui

The Network Time Protocol (NTP)

- NTP computers synchronise with one another using one of the following modes:
  - ***multicast mode***
    - intended for use in high speed LANs with low accuracy required.
    - one server periodically multicasts the time to other servers on LAN
    - which sets their clocks assuming a small delay.

The Network Time Protocol (NTP)

- ***Procedure call mode***
  - - similar to Cristian's method and give better accuracy.

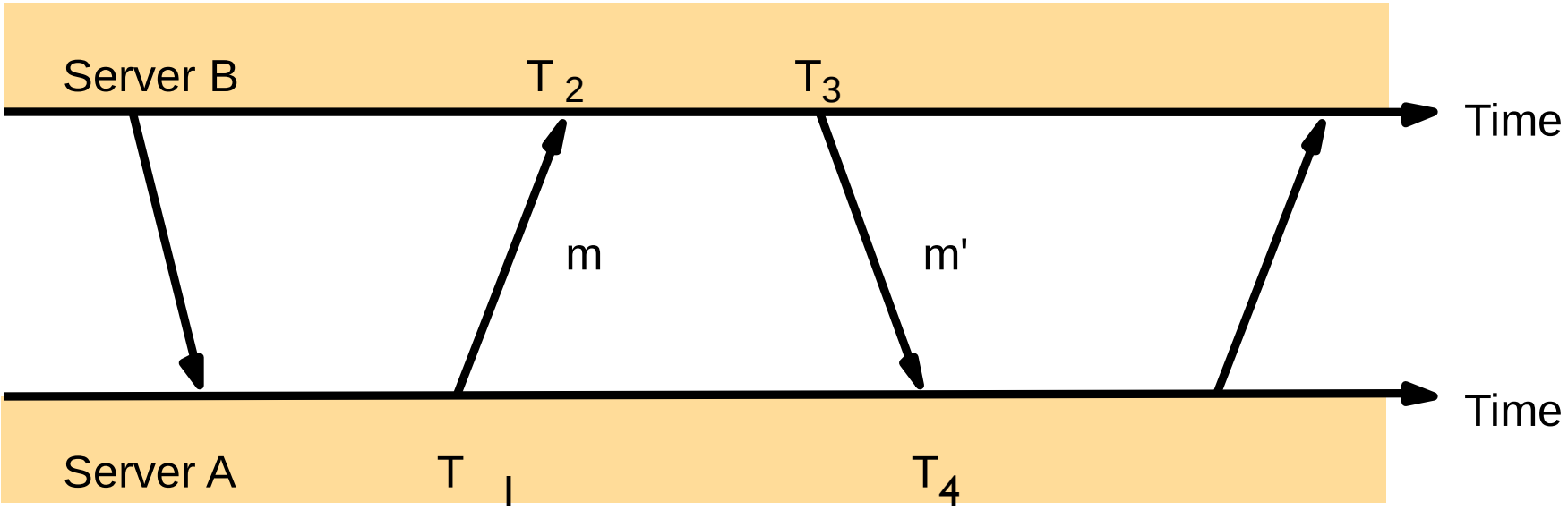
- ***Symmetric mode***
  - - used where the highest accuracy needed.
  - - servers exchange time info as part of info exchange.

- All modes use UDP transport protocol.

In *procedure-call mode* and *symmetric mode*, processes exchange pairs of messages. Each message bears **three** timestamps from recent message events:

- local times when previous NTP message between the pair was sent/received, T1, T2.
- local time when current NTP message was transmitted, T3.

The recipient of the NTP message notes local time, T4, when message was received.



Note that unlike Cristian's algorithm, there is a non-zero delay between the arrival of one message and the dispatch of next.

For each *pair of messages* sent between two servers, the NTP calculates:

- an offset O_i which is an estimate of the actual offset between the two clocks
- a delay d_i which is the total transmission time for the two messages.

If the true offset of the clock at B relative to the clock at A is O and the actual transmission time for messages m_1 and m_2 are t_1 and t_2 respectively, then

$$T_2 = T_1 + t_1 + O$$

$$T_4 = T_3 + t_2 - O$$

This gives

$$d_i = t_1 + t_2 = (T_2 - T_1) + (T_4 - T_3)$$

Also

$$O = O_i + (t_2 - t_1)/2 \text{ where } O_i = ((T_2 - T_1) + (T_3 - T_4))/2$$

Since $t_1, t_2 >= 0$, it can be shown that

$$(O_i - d_i/2) <= O <= (O_i + d_i/2)$$

Thus O_i is an estimate of offset and d_i is a measure of accuracy of this estimate.

Exercises

An NTP server B receives server A's message at 16:34:23.480 bearing a timestamp 16:34:13.430 and replies to it. A receives the message at 16:34:15.725, bearing B's timestamp 16:34:25.7. Estimate the offset between B and A and the accuracy of the estimate.

The Network Time Protocol (NTP)

- NTP servers apply a data filtering algorithm to successive pairs $<O_i, d_i>$ which estimates the offset O and calculates the quantity of this estimate as a statistical quantity called filter dispersion.

- A relatively high dispersion represents unreliable data. The eight most recent pairs $<O_i, d_i>$ are retained and the value of O_j that corresponds to the minimum value of d_i is chosen to estimate O.

The Network Time Protocol (NTP)

- In general, an NTP server engages in message exchanges with several of its peers. NTP servers also run peer-selection algorithms to eliminate servers with unreliable values.

- Peers with lower stratum numbers are favored over higher strata since they are closer to primary time sources.

The Network Time Protocol (NTP)

- NTP employs a phase lock loop model (Mills 1995) which modifies the local clock's update frequency in accordance with observations of its drift rate. For example, if a clock is discovered always to gain time, say at 6 seconds per hour, then its frequency is reduced slightly (in software or hardware) to compensate for this.

- The clock's drift in the intervals between synchronization is thus reduced. Mills quotes synchronous accuracy's in order of 10s ms over Internet paths and 1ms over LANs.

The Network Time Protocol (NTP)

- The degree of synchronisation accuracy that is practically obtainable fulfils many requirements but is nonetheless not sufficient to determine *the ordering of an arbitrary pair of events* occurring at different computers.

- The ***happened-before*** relation is a partial order on events that reflects a flow of information between them - within a process or via messages between processes.

The Network Time Protocol (NTP)

- Some algorithms require events to be ordered in *happened-before* order, for example successive updates made at separate copies of data.

- **Lamport clocks** are counters that are updated in accordance with the *happened-before* relationship between events.

- **Vector clocks** are an improvement on Lamport clocks, because it is possible to determine by examining their vector timestamps whether two events are ordered by *happened-before* or are concurrent.

Some Final Points

- The lack of universal time is a fundamental characteristic of distributed systems. However, it is possible to synchronise clocks to reasonable degrees of accuracy, which are adequate for many purposes.

- Internal synchronisation algorithms synchronise clocks together, without reference to UTC; external synchronisation algorithms synchronise them to UTC.

Some Final Points

- Where clock accuracy is an important requirement rather than a convenience (i.e. *real-time distributed systems*), a more critical analysis is required than that given in this lecture - in particular where clocks may report false values, and where time servers must be assumed to fail occasionally.

- Questions then arise about the guarantees that can be placed on clock values returned to the user.

Some Final Points

- Logical clocks enable us to reason about the order in which events occur without reference to real time.

- The *happened-before* relation captures events that are related at the level of the application semantics; unfortunately, it can also capture events that are unrelated at this level.