Today's Topics

Chapter 10.

- Clocks
  - Physical Clocks.
  - Synchronising physical clocks
- Logical Clocks

What is time?

”What, then, is time? If no one asks me, I know what it is.
If I wish to explain it to him who asks me, I do not know.”

St. Augustine, Confessions Bk 11, Ch XIV

What is time?

- How do you define a second?
  - The answer depends on how accurate you want to be. You could define a second to be a 60th of a minute and a minute to be a 60th of an hour and there to be 24 hours in a day. (Easy to work out how long a day). In fact there are ways of working out the length of the mean solar day, by solar observations.
  - Is was discovered in the '40s that the period of the earth's rotation is not constant. The earth is slowing down due to tidal friction and atmospheric drag. It is believed that 300 million years ago there where about 400 days per a year.

The Atomic Clock

- In 1948 it the atomic clock was invented and a second was defined as 9,192,631,770 transitions of a cesium 133 atom. The choice of 9,192,631,770 was made to make the atomic second equal to the mean solar second in the year of its introduction.
- An atomic clock counts how many ticks of a cesium atom have been made since midnight Jan 1, 1958. This is TAI (International Atomic Time as defined by the Bureau International de l'Heure in Paris).

Leap Seconds

.

- But now a TAI day 86400 TAI seconds is about 3 msec longer than a mean solar day.
- If this carried the calender would be out of phase with the days.
- So to avoid problems leap seconds are introduced whenever the discrepancy between TAI and solar time grows to 800 msec. The corrected time is called Universal Coordinated Time, UTC.
- By January 1999, 29 leap seconds had been introduced.

Real Clocks

- Let t be the real time given a clock A let $C_A(t)$ be the time clock A reads at time t.
- If the clock was perfect the for all t then $C_A(t) = t$
- But a real clock there is some clock drift and there is some constant ρ such that:

$$1 - \rho \leq \frac{dC}{dt} \leq 1 + \rho$$

- For clocks based on quartz crystals ρ is about 10^{-6} seconds/seconds given about 1 second difference every 11.6 days.

When to Synchronise Clocks?

- Given two clocks A and B with drift ρ how often should we synchronise them so that the difference is no more than δ?
- Suppose that the worst thing happens:

$$\frac{dC_A(t)}{t} = 1 - \rho \text{ and } \frac{dC_B(t)}{t} = 1 + \rho$$

- Assume that at time 0, $C_A(0) = C_B(0) = 0$ after t seconds $C_A(t) = (1 - \rho)t$ and $C_B(t) = (1 + \rho)t$ so the difference is:

$$C_B(t) - C_A(t) = t(1 + \rho - (1 - \rho)) = 2\rho t$$

  and we want $\delta < 2\rho t$ so we need to synchronise at least every $\delta/(2\rho)$ seconds.

Why Bother to Synchronise clocks

- Many applications depend on timestamps.
- For example the unix make system will compare the timestamp of a source file *hello.c* with its object code *hello.o* if the source file is older than the the object code then the file has to be recompiled.
- If the compiler is running on a different machine to the editor and the clocks generating the timestamps of the file where out of phase then bad things might happen.

Clock Synchronisation Algorithms

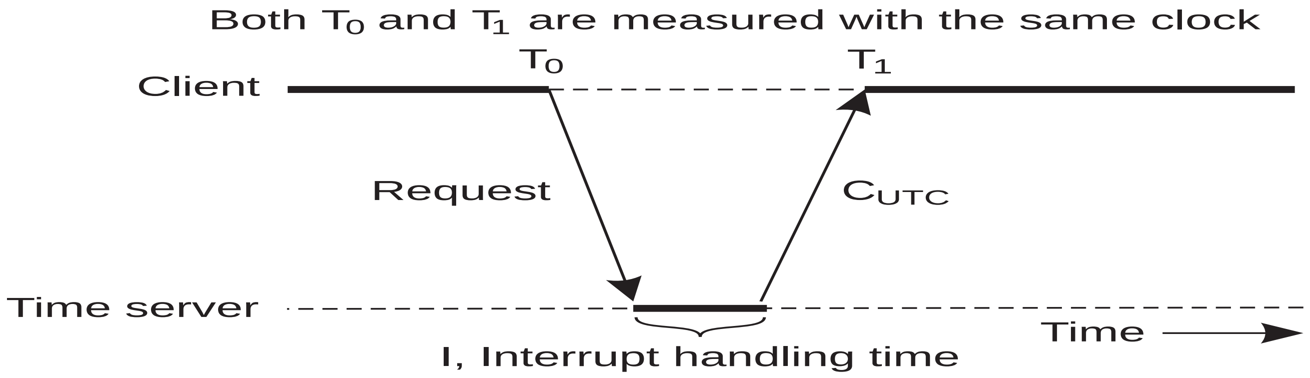
- Cristian's Algorithm: Poll a central server, estimate the round trip time.
- The Berkeley Algorithm: Distributed, try to find a common notion of time.
- The Network Time Protocol, practical method to synchronise clocks in a network.

What is the problem?

- Network delay is unbounded and unpredictable.
- If you ask the time the answer you get back is out of date by the time it gets to you.
- The best you can do is produce an algorithm that within a certain probability will synchronise the clocks within a certain δ.

Cristian's Algorithm

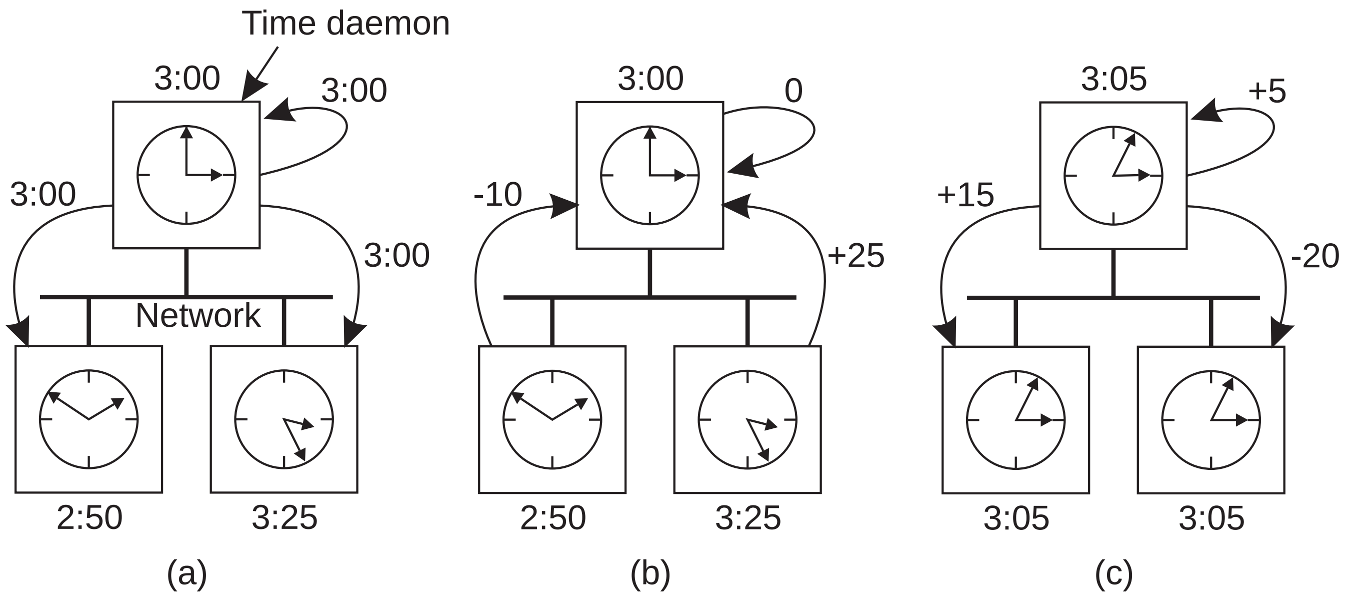
- Basic architecture. Client and Server. The server holds the correct time.
- Client request the time from server, server replies with the time. The Client tries to calculate the round-trip time.


Both T_0 and T_1 are measured with the same clock
T_0
T_1
Client
Request
C_{UTC}
Time server
Time
I, Interrupt handling time

Cristian's Algorithm

- The Client process measures the round-trip time using its internal time. Assumption Round trip time is of a larger enough order of magnitude so that clock drift does not matter.
- Assume that the outgoing and incoming messages are roughly the same.
- So the propagation delay is then $(T_1 - T_0 - I)/2$ this can be then used to set the clock of the client together with the time sent back from the server.

The Berkeley Algorithm



The Berkely Algorithm

- One process is designated the master.
- The master periodically polls all the slaves for their times.
- Round-trip times are estimated as in Cristian's algorithm.
- The master process averages all the times and sends out new corrections.
- On average differences are cancelled out and the clocks converge to a common time.

The Berkely Algorithm

- If the maximum round-trip time, T_M, is know (or if the master discard messages with a round-trip time longer than T_M) then the minimal possible transmission time between two nodes can be calculated:
$$\epsilon = \frac{T_M - 2\min(T_{AB}, T_{BA})}{2}$$
where T_{AB} is the minimum transmission time from A to B and T_{BA} is the minimum transmission time from B to A.
- Then it can be shown that if you synchronise every T seconds then the time of all non-faulty clocks will be within the range $4\epsilon + 2\rho T$ ($\rho =$ clock drift).

Lamport Time Stamps

A quote from Lamport's original paper[a]

> The concept of time is fundamental to our way of thinking. It is derived from the more basic concept of order in which events occur. We say that something happened at 3:15 if it occurred after our clock read 3:15 and before it read 3:15. The concept of the temporal ordering of event pervades our thinking about systems. For example in an airline reservation system we specify that a request for a reservation should be granted if it made before the flights filled. However, we will see that this concept must be carefully reexamined when considering events in a distributed system.

[a] "Time, Clocks, and the Ordering of Events in a Distributed System", Leslie Lamport. Communications of the ACM 1978, VOl. 21 No.7 558-565

Lamport Time Stamps

- In a distributed system network delays are unbounded.
- Two travel agents booking a flight at the same time, the server does not know which one was sent first.
- Lamport timestamps try to characterise the notion of happens before. But it no longer a total order, but a partial order. There are some events that you don't know which order they happened in.

Lamport Timestamps

- Let $\rightarrow$ be the happens before relation, $a \rightarrow b$ reads that a happens before b.
- $\rightarrow$ has to obey some axioms:
  - For all events it should be false that $a \rightarrow a$.
  - If a happens before b on the same processor then $a \rightarrow b$.
  - $a \rightarrow b$ and $b \rightarrow c$ implies $a \rightarrow c$.
  - If a is the event of sending a message and b is the event of receiving that message then $a \rightarrow b$.

Lamport Timestamps

- A logical clock is a mapping L from events to the natural numbers such that:

$$a \rightarrow b \Rightarrow L(a) < L(b)$$

- It does not mean that if $L(a) < L(b)$ then $a \rightarrow b$.

Lamport Timestamps

Lamport's logical clock is quite simple it uses the following three

rules to give a stamp to each event in the system. Each process, i, has its own counter L_i which is maintained as follows:

1. When ever an event happens in process i, set L_i to $L_i + 1$.
2. A process p_i sends a message m, it adds a timestamp to m L_i.
3. When process j receives a message (m, L_i), L_j is set to $\max(L_j, L_i)$ ands then applies the first rule before timestamping the receive event.

Lamport Timestamps

$$\begin{tikzcd}
p_1 \arrow[r, dash] & a_1 \arrow[r, dash] & b_2 \arrow[rrrr] \arrow[dr, "{(b,2)}"] & & & & {} \\
p_2 \arrow[rrr, dash] & & & c_3 \arrow[r, dash] & d_4 \arrow[rr] \arrow[dr, "{(b,4)}"] & & {} \\
p_3 \arrow[r, dash] & e_1 \arrow[rrrr, dash] & & & & f_5 \arrow[r] & {}
\end{tikzcd}$$

In this example $a \rightarrow b$, $b \rightarrow c$, $c \rightarrow d$, $e \rightarrow f$ and $d \rightarrow f$.

But not $e \rightarrow a$ or $a \rightarrow e$.

Lamport Timestamps

- Lamport timestamps put a total order on a partial order.
- It is a total order that every process can agree on.
- The total order can be used to decide on the ordering or requests.
- Of course it does not really tell you which happened first, but it gives you an order than every body can agree on.