Clock Synchronization - Disposition

Lukas Peter Jørgensen, 201206057, DA4

26. marts 2014

1 Physical time

1.1 Problems

Not all computers can have precise atomic clocks since they are expensive.
Internal time deviates between computers, some tick faster, some tick slower.

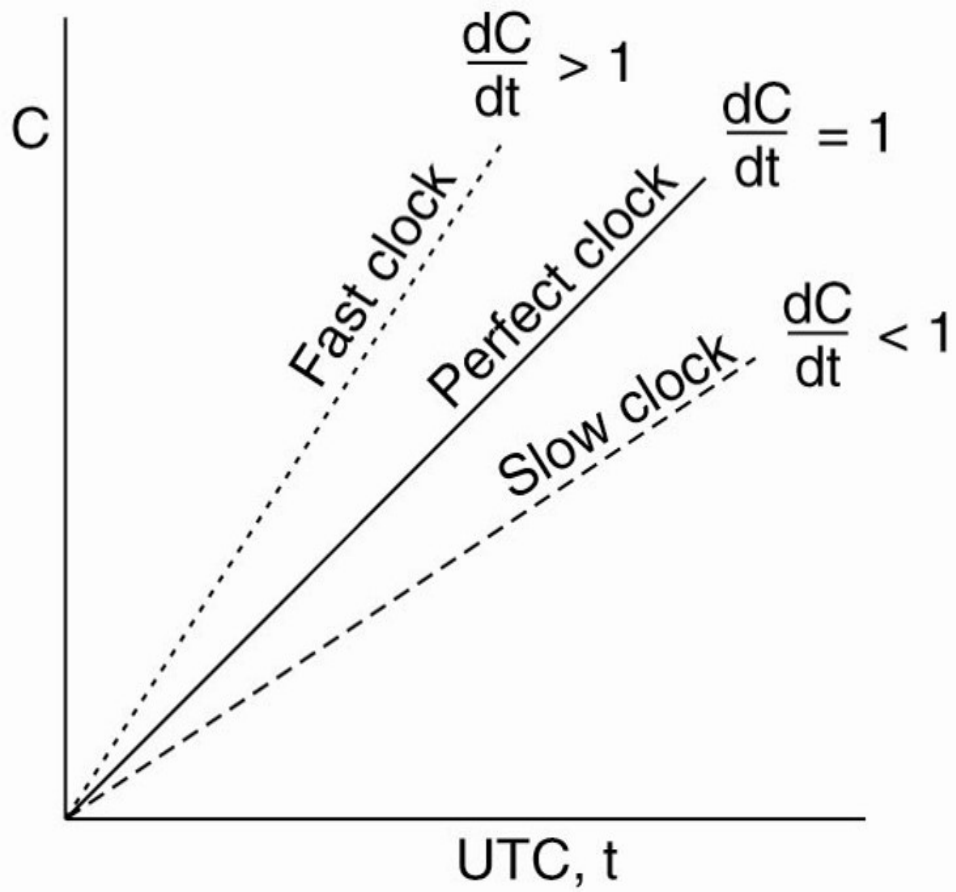
1.2 Bad solutions

Listen for radio transmitters (WWV), but accuracy is 10msec in which time somewhere around 30.000.000 instructions can be executed on a 3GHz computer.
Listen for GPS satellites broadcasts their position and time, but accuracy is 60nsec in which 180 instructions can be executed on a 3GHz computer, and it needs a good view to the sky as well, which computers seldomly has.

1.3 Better solution: Network Time Protocol

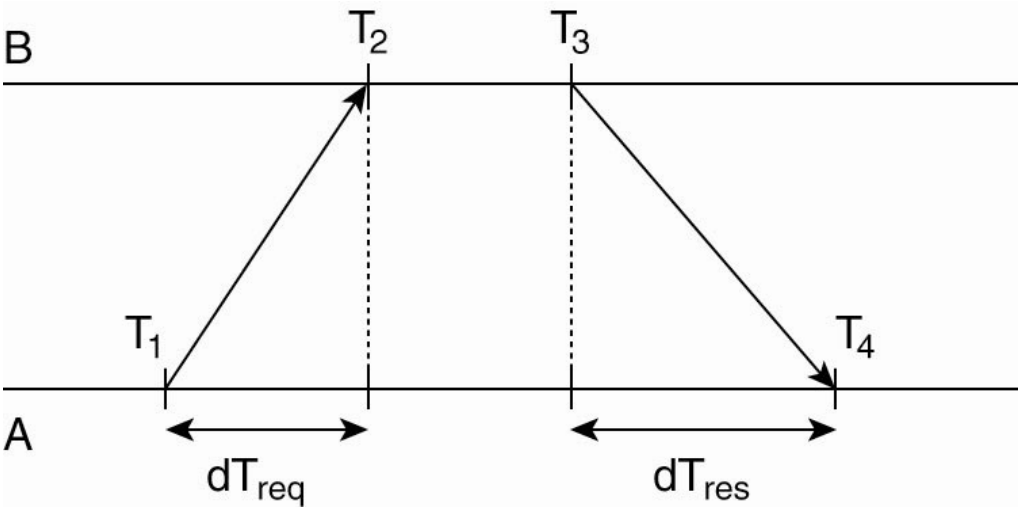
Let clients contact a time server with a WWV receiver or an accurate clock, which can accurately provide the current time. The issue is that message delays will have outdated the reported time, the trick is to find a good estimate for these delays.

Clock skew



Internal clock on the y-axis, UTC on the x-axis.
$\frac{dC}{dt}$ = the internal clock divided by the external clock.

2 Christian's algorithm



A's offset relative to B is $\Theta t_B - t_A$

$$\Theta = T_2 - (T_1 + dT_{req}) = T_2 - T_1 - dT_{req}$$

$$\Theta = T_3 - (T_4 - dT_{res}) = T_3 - T_4 + dT_{res}$$

$$2\Theta = (T_2 - T_1) + (T_3 - T_4) + (dT_{res} - dT_{req})$$

$$\approx (T_2 - T_1) + (T_3 - T_4)$$

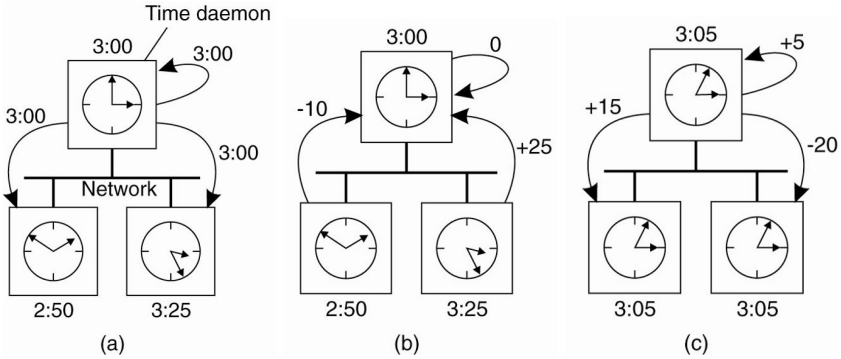
$$t_B - t_A \approx \frac{(T_2 - T_1) + (T_3 - T_4)}{2}$$

Likewise we can calculate the delay (an estimate for dt_req and dT_res) as:

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

Do this a number of times and pick Θ where the delay, i.e. δ, is smallest.

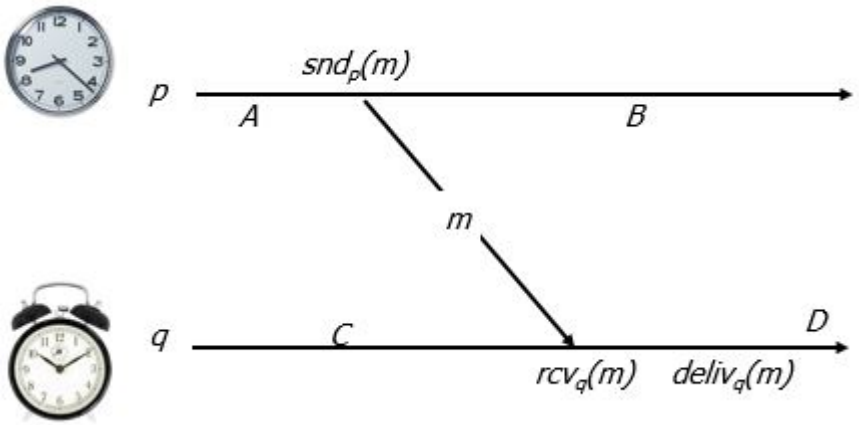
3 Berkeley algorithm



4 Logical time

When you just need a well-defined ordering of events and not physical time. For example an online-forum, where we want all answers to be posted after the questions and we want all answers to be in the same order on all replicas of the forum.
That is, all we care about is whether event A *happened before* event B.

4.1 The happens-before relation



Local ordering

A happens before $snd_p(m)$ which happens before B, and C before $rcv_q(m)$ which happens before D.
This is the local ordering, at each individual process, denoted:

$$A \xrightarrow{p} snd_p(m) \xrightarrow{p} B$$

And

$$C \xrightarrow{q} rcv_q(m) \xrightarrow{q} D$$

Distributed ordering

$snd_p(m)$ happens before $rcv_q(m)$, this is the distributed ordering of the system, denoted:

$$snd_p(m) \xrightarrow{M} rcv_q(m)$$

Transitive ordering

Since $A \rightarrow snd_p(m)$ and $snd_p \rightarrow rcv_q(m)$ we get that $A \rightarrow rcv_q(m)$ and therefore $A \rightarrow D$.

Concurrent ordering

B and D are *concurrent*, even though it looks like B happens before D, there is know way of knowing since no information have flowed between the two processes.

4.2 Logical clocks

Each process holds a local counter C_p (integer) and everytime an event that matters to p happens at a process p, the process increments C_p.
When p sends m it sets:

$$C_m = C_p$$

When q receives m set:

$$C_q = max(C_q, C_m) + 1$$

From previous image:

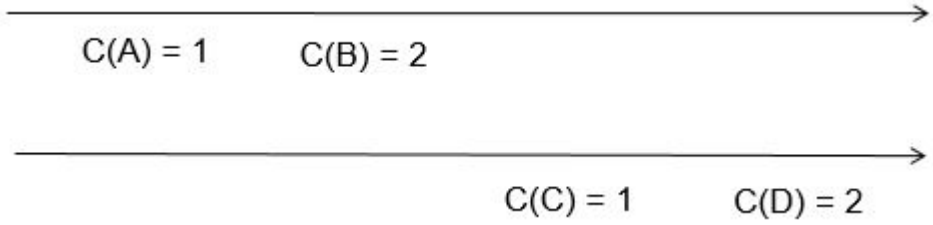
$$C(A) = 1, C(snd_p(m)) = 2, C(m) = 2, C(B) = 3$$

$$C(C) = 1, C(rcv_q(m)) = max(1, 2) + 1 = 3, C(deliver_q(m)) = 4, C(D) = 5$$

Comparison with Happens-Before

If $A \xrightarrow{p} B$ then $C(A) < C(B)$
If $A \xrightarrow{M} B$ then $C(B) = C(A) + 1$
If not $A \xrightarrow{p} B$ or $A \xrightarrow{M} B$, and there exist C such that $A \rightarrow C \rightarrow B$; induction says $C(A) < C(C)$ and $C(C) < C(B)$, so $C(A) < C(B)$.

But, $C(A) < C(B) \implies A \rightarrow B$.
Because, processes that don't communicate still assign timestamps and hence events will "seem"to have an order.
For example:



So $C(C) < C(B)$ but we don't have $C \to B$.