Disposition 8: Synchrony

Mathias Ravn Tversted

January 6, 2020

Table of contents

First slide

- Clocks
- Known drift
- delivery times

Physical Time

Computers need access to physical time. Most consumer grade computers have quartz crystal clocks. These drift by about 1 second every 10 days. They drift 2^{-20} seconds per second. A modern CPU can execute 1000 instructions in that time. It is therefore a long time. There are also atomic clocks, that lose 1 second per 1 million years.

GPS Synchronisation

Global Positioning System has large number of satellites in low orbit with atomic clocks. They transmit their position and time. Position is (x, y, z) giving 4-dimensions (x, y, z, t). If you receive 4 signals, then you get 4 equations with 4 unknowns, and thus you can compute your own position.

NTP Clock Synchronisation

NTP stands for *Network Time Protocol*. Let S be a server with an atomic clock or something close to it. Let C be a client with a drifting clock. We have the following two assumptions:

- During the running of the protocol, the drift is negligible
- The time it takes to send from the client is the same as from the server to the client

If the entire protocol takes a second, and the client has a quartz clock, it might only drift by 2^{-20} seconds which is fair.

NTP Protocol

- C sends "time request" message to S and stores its current system time T_1
- Upon receiving the "time request" message, the server S stores its current system time T_2
- The server S prepares a "time response" message, which includes the time T_2. Right before the response to the client C, it computes T_3 and sends it all.
- Upon receiving T_2, T_3, the client C measures current system time T_4.
- Compute $TransEst = \frac{(T_4 - T_1) - (T_3 - T_2)}{2}$
- Compute $OffsetEst = (T_1 + TransEst) - T_2$

NTP Protocol: Why does it work?

The clock of the client initially is $Offset = T_C - T_S$ ahead. Now $T_S = T_C - Offset$. Let $Trans$ be the transport time. Then, when server measures T_2, the clients time is $T_1 + Trans$. The time at the client is $T_1 + Trans$. Offset could then be $Offset = (T_1 + Trans) - T_2$. Transmission time cannot be computed accurately, because they the two clocks are not synchronised. The way to fix this, realise that $T_4 - T_1$ is the total time it took to run the protocol. T_1, T_4 are run on the same clock, so this is fine. Time spent on server side $T_3 - T_2$ is also well defined. Therefore, the total time spent sending both messages is $(T_4 - T_1) - (T_3 - T_2)$. The clocks don't drift noticably, and the time it takes to send is

Adjusting the Clock and assumptions

Instead of jumping backwards and forwards, which can disturb processes, we instead speed the time up or slow it down in order to peacefully synchronise the time.

The assumption of transfer times is optimistic, since there many be many variations in network transfer times. This is why the NTP protocol runs several times and adopts the one where the transfer estimate is lowest, because the ones that have the highest transfer time may also be the ones where errors occur (???)

If one knows the bound on network delay, and how much clocks can drift, it is possible to compute the *Max Clock Drift* when occasionally running NTP.

The fully synchronous Round-based protocols

For the fully synchronous Round-based model to work, consider n parties or processes $P_1, \ldots, P_n$. The protocol, π, proceds in *rounds*. In each round, one can send a message or NoMsg. Assume that all parties have perfectly synchronised clocks, messages arrive in the following round and that transmission time is fixed.

Message arrival

The assumptions are not entirely realistic, but we can still hope to set bounds on drift and delivery times. If party knows that someone will send a message at time t. If it knows Offset, Trans. If it receives nothing at $t + 2\mathit{Offset} + \mathit{Trans}$ then it knows that nothing was sent. At real time, $(t + \mathit{Offset} + \mathit{Bound}) + \mathit{Offset} = t + 2\mathit{Offset} + \mathit{Trans}$

Accounting for Computation Time

We can now set timeouts so that we do not drop messages that arrive too late. Let *MaxComp* be the maximum time it takes for any party to complete the necessary computation. We also assume a positive bound on *MaxTrans* and one on *MaxDrift*, which can be kept down with clock synchronisation. Let now $\mathit{SlotLength} = 2\mathit{MaxDrift} + \mathit{MaxTrans} + \mathit{Maxcomp}$. This ensures that all honest parties have time to compute and send messaes.
Let t_0 be the time everyone agreed to start the protocol. The input to P_i is (t_0, x_i). All honest parties agree on t_0. We assume that this arrives at t_0.