Distributed systems – theory
7. Algs for clock synchronization

Goals and assumptions

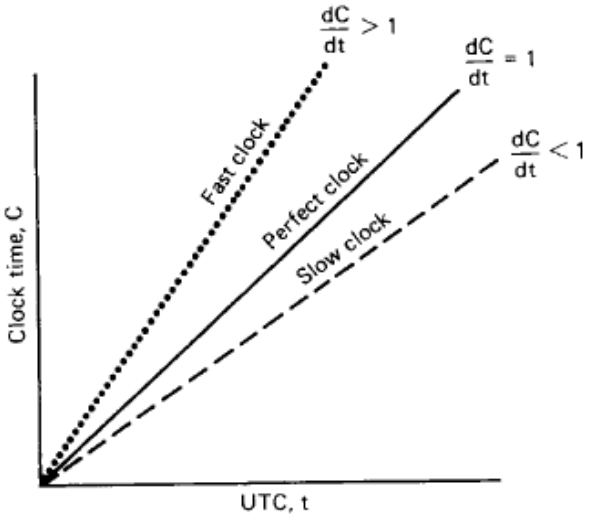
- Ass. 1: one machine has a UTC receiver,
  - Goal: keeping all the other machines synchronized to it.
- Ass. 1': machines have UTC receivers, each machine keeps track of its own time
  - Goal: keep all the machines together as well as possible
- Ass. 2: Each machine is assumed to have a timer that causes an interrupt *H* times a second.
  - When this timer goes off, the interrupt handler adds 1 to a *software clock* that keeps track of the number of ticks (interrupts) since some agreed-upon time in the past.
  - Let us call the value of this clock *C*.
  - When the UTC time is *t*, the value of the clock on machine *p* is $C_p(t)$.
  - In a perfect world, we would have $C_p(t) = t$ for all *p* and all *t*. In other words, d*C*/d*t* ideally should be 1.

Drift rate

- Real timers do not interrupt exactly *H* times a second.
  - Theoretically, a timer with *H* = 60 should generate 216,000 ticks per hour.
  - In practice, the relative error obtainable with modem timer chips is about meaning that a particular machine can get a value in the range 215,998 to 216,002 ticks per hour.
- If there exists some constant such that

  $1-r <= dC/dt < 1+r$

  the timer can be said to be working within its specification.
- The constant *r* is specified by the manufacturer and is known as the maximum *drift rate*.

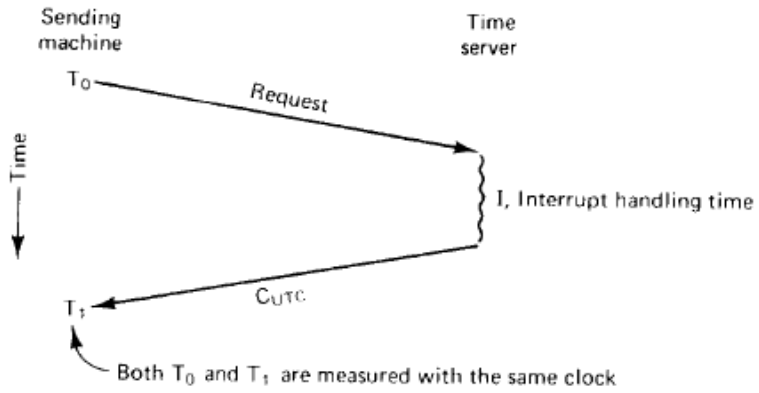


Synchronization

- If two clocks are drifting from UTC in the opposite direction, at a time d*t* after they were synchronized, they may be as much as 2 *r* d*t*.
- If the DS designers want to guarantee that no two clocks ever differ by more than *d*, clocks must be resynchronized (in software) at least every *d/2r* seconds.
- The various algorithms differ in precisely how this resynchronization is done.

Cristian's Algorithm – principle

- Assume: DS in which one machine has a UTC receiver
  - Let us call the machine with the receiver a *time server.*
- Goal is to have all the other machines stay synchronized with it.
- Periodically, certainly no more than every *d/2r* seconds, each machine sends a message to the time server asking it for the current time.
- The time server C_{UTC} as fast as it can with a message containing its current time.
- When the sender gets the reply, it can just set its clock to C_{UTC}.



Cristian's Algorithm – problems

- The algorithm has two problems:
  - major problem: is that time must never run backward
    - if the sender's clock is fast, will be smaller than the sender's current value of *C*.
    - e.g. problems: an object file compiled just after the clock change having a time earlier than the source which was modified just before the clock change.
  - minor problem: is that it takes a nonzero amount of time for the time server's reply to get back to the sender.
    - this delay may be large and vary with the network load.

Dealing with the major problem

- The change must be introduced gradually.
- One way is as follows:
  - Suppose that the timer is set to generate 100 interrupts per second.
  - Normally, each interrupt would add 10 msec to the time.
  - When slowing down, the interrupt routine adds only 9 msec each time, until the correction has been made.
  - Similarly, the clock can be advanced gradually by adding 11 msec at each interrupt instead of jumping it forward all at once.

Dealing with the minor problem

- Cristian's way of dealing: attempt to measure the network delay.
- The sender record accurately the interval between sending the request to the time server and the arrival of the reply.
- The starting time, *T*0 and the ending time, *T*1, are measured using the same clock,
  - the interval will be relatively accurate, even if the sender's clock is off from UTC by a substantial amount.
- In the absence of any other the best estimate of the message propagation time is (*T*1-*T*0)/2.
- When the reply comes in, the value in the message can be increased by this amount to give an estimate of the server's current time.
- If the theoretical minimum propagation time is known, other properties of the time estimate can be calculated.

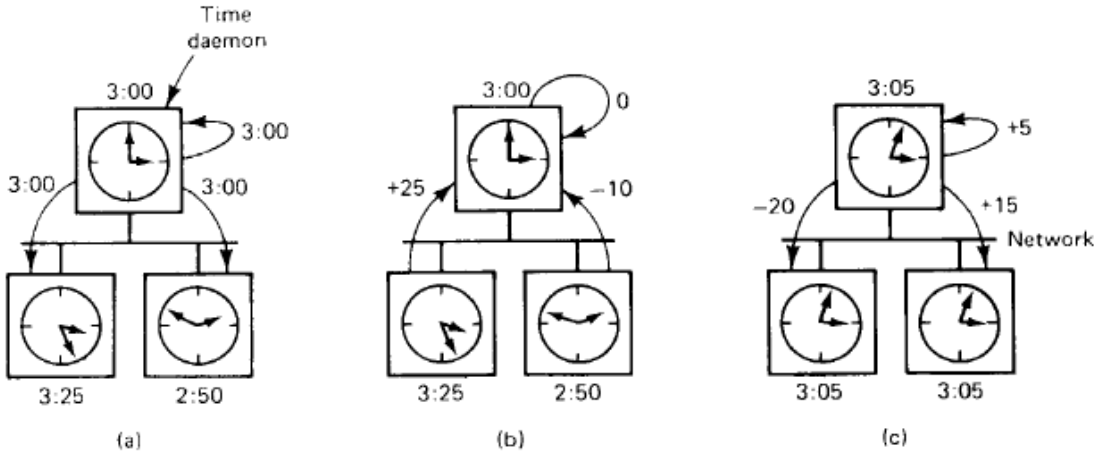
Improving the delay estimation

- The estimate can be improved if it is known approximately how long it takes the time server to handle the interrupt and process the incoming message.
- Let us call the interrupt handling time *I*.
- The amount of the interval from *T*0 to *T*1 that was devoted to message propagation is *T*1- *T*0-*I* so the best estimate of the one-way propagation time is half this.
- But … Systs do exist in which messages from A to B systematically take a different route than messages from B to A, and thus have a different propagation time.
- To improve the accuracy, Cristian suggested making not one measurement, but a series of them.
  - Any measurements in which *T*1- *T*0 exceeds some threshold value are discarded as being victims of network congestion and thus unreliable.
  - The estimates derived from the remaining probes can then be averaged to get a better value.
  - Alternatively, the message that came back fastest can be taken to be the most accurate since it presumably encountered the least traffic underway and thus is the most representative of the pure propagation time.

Berkeley Algorithm (1/2)

- Cristian's algorithm: the time server is passive.
  - Other machines ask it for the time periodically.
  - All it does is respond to their queries.
- Berkeley algorithm: the opposite approach is taken
  - Here the time server (actually, a time daemon) is active, polling every machine periodically to ask what time it is there.
  - Based on the answers, it computes an average time and tells all the other machines to advance their clocks to the new time or slow their clocks down until some specified reduction has been achieved.
  - This method is suitable for a system in which no machine has a UTC receiver.
  - The daemon's time must be set manually by the operator periodically.

Berkeley Algorithm (2/2)



a) At 3:00, the time daemon tells the other machines its time and asks for theirs.
b) They respond with how far ahead or behind the time daemon they are.
c) Armed with these numbers, the time daemon computes the average and tells each machine how to adjust its clock

Averaging Algorithms

- Both methods described above are centralized, with the usual disadvant.
- Decentralized algorithms are also known!
- One class of decentralized clock synchronization algorithms works by dividing time into fixed-length resynchronization intervals.
  - The *i*th interval starts at $T0+iR$ and runs until $T0+(i+1)R$ where $T0$ is an agreed upon moment in the past, and R is a system parameter.
  - At the beginning of each interval, every machine broadcasts the current time according to its clock.
  - Because the clocks on different machines do not run at exactly the same speed, these broadcasts will not happen precisely simultaneously.
  - After a machine broadcasts its time, it starts a local timer to collect all other broadcasts that arrive during some interval S.
  - All the broadcasts arrive, thencompute a new time from them
    1. The simplest algorithm is just to average the values from all the other machines.
    2. A slight variation on this theme is first to discard the m highest and m lowest values, and average the rest
    3. Another variation is to try to correct each message by adding to it an estimate of the propagation time from the source.

Multiple External Time Sources

- For systems in which extremely accurate synchronization with UTC is required, it is possible to equip the system with multiple receivers
- Due to inherent inaccuracy in the time source itself + fluctuations in the signal path, the best the DS can do is establish a range (time interval) in which UTC falls.
- In general, the various time sources will produce different ranges, which requires the machines attached to them to come to agreement.
- To reach this agreement, each processor with a UTC source can broadcast its range periodically, eg at precise start of each UTC min.
- None of the processors will get the time packets instantaneously.
- Worse yet, the delay between transmission and reception depends on the cable distance and no. of gateways that the packets have to traverse, which is different for each (UTC source, processor) pair.
- Other factors can also play a role, such as delays due to collisions when multiple machines try to transmit simultaneous on a network.
- If a processor is busy handling a previous packet, it may not even look at the time packet for a considerable number of milliseconds, introducing additional uncertainty into the time.