CSE P 590 SG Assignment #2a Sample Solution

Andy Collins

CSETP—The CSE Time Protocol using TCP

Questions

1. Measure the latency between your test machine and tioga. What is a reasonable absolute minimum RTT to assume? Note that the CSETP server on tioga adds a minimum 20ms one-way delay (each way) to each message. This is in addition to the minimum network delay. You can either measure the total delay by observing the CSETP messages, or you can use `ping` to measure the network delay and add the 20ms.

   Using this value, and knowing that the ρ for the master server is no worse than 0.01, what is the best precision we can hope to achieve in synchronizing the clock (meaning for a single experiment)?

   Answer The min RTT depends, of course, on the path being measured. Within our building, it is so small that it is pointless to use `ping` to measure it (and equally pointless to use Java timing to measure it). So for myself, I did the calculations assuming *zero* additional delay beyond the 40ms. From an engineering point of view, this is the best solution unless the true latency is so large that you can't reasonably make the syncronization spec. Hopefully in this case the latencies will be both large enough to measure and real enough to make the measurements and the assumption that one-way delay equals half-RTT true.

   For 20ms min one-way delay, and assuming the client clock drifts as badly as the server clock, we can use equation 11 from Cristian's paper to get:

   $$e_{min} = 3 \cdot \rho \cdot min = 3 \cdot 0.01 \cdot 20ms = 600\mu s$$

   This is surprisingly small, but correct. The point is that this is the absolute best syncronization possible, and we will in fact achieve rapport with probability approaching zero if we try this.

2. Devise a maximum allowable RTT and a minimum resynchronization period (respectively the largest RTT query/response that can be used to synchronize the clock and the time between resynchronizations) that will ensure that your clock tracks tioga's CSETP time with no more than a 200ms error, based on the observed typical delays and the known ρ. About what fraction of synchronization attempts do you expect will succeed?

   Recall from [Cri89] that this is a fundamental tradeoff: the smaller the max RTT, the more accurate the synchronization but the more likely we will fail to synchronize and have to try again. But the more accurate the synchronization, the longer we can let the clock run before we have to resynchronize.

   Answer I chose to set the per-episode accuracy at 150ms, under the theory that I'd rather have this be relatively loose, and therefore have a high likelihood of successful rapport, than have a particularly long period between rapports. From this, we derive the max RTT as

   $$2U = 2(1 - 2\rho)(150ms - 20) = 255ms$$

again assuming all clocks are as bad as $\rho = 0.01$, and no additional min delay beyond the 20ms. I then figured the max time between rapports (simplified) by asking how long a clock drifting at $\rho = 0.01$ would take to move from a 150ms error to 200ms error:

$$T = \frac{(200ms - 150ms)}{2\rho} = 2500ms$$

This is based on a simple worst-case assumption: that the rapport error is maximal and in the direction of drift, and that the total effective drift is 2ρ because the two clocks are moving in opposite directions (i.e. my clock is 150ms later than authoritative time, and my clock is running fast and authoritative time is running slow). In principle, we should account for the part of this that is taken up with rapport, but I didn't bother because I knew that this would be small (my min RTT is large enough that multiple attempts are rare) and because the both-bad-clocks assumption means that I've really got a factor of two slop to play with.

Firming up the probability of successful rapport means looking at the delays seen inside the protocol. I used the test engine to do so with a nearby client, and got the following 20 values, all in milliseconds:

113	119	199	139	129	159	99	109	159	219
289	129	329	89	129	79	99	19	129	219

So from this small sample, 2 out of 20 attempts would have failed. If we take $P = 0.9$ as the probability of success, then we might choose to fix $k = 5$ to achieve 99.999% probability of success. We don't need to wait between retries, because this delay is already independent one request to the next. So we could apply equation 16 to get that the max rapport spacing is

$$T = \frac{1 - \rho}{\rho} 200ms - 5 \cdot 500ms = 17.3sec$$

where 500ms is a made-up number for the maximum RTT. This, of course, would only be allowable if we achieved a rapport accuracy close to the best possible ($600\mu s$, from question 1). Applying equation 17, we get the min rapport spacing, assuming $U = 128ms$, of

$$T = \frac{1 - \rho}{\rho}(200ms + 20ms - 128ms) - 5 \cdot 500ms = 6.6sec$$

Both of these are, of course, based on Cristian's assumption that $\rho \cdot RTT$ is negligible, which is only barely reasonable for us, although I said you could use it.

Code

The code for my solution and authoritative server in one, written in Java, is posted separately on the course web page. The solution itself is in one file, broken into several classes. The description and roadmap is in the comments at the top. My solution uses the `OffsetClock` provided with the assignment.

References

[Cri89] Flaviu Cristian. A Probabilistic Approach to Distributed Clock Synchronization. In *Proceedings of the 9th International Conference on Distributed Computing Systems (ICDCS)*, pages 288–296, June 1989.