Computer Science 2221 **Homework 2 Answers**

Behdad Esfahbod **Student # 993505827**

---

1. (a) We have two processes s and p and q, and $t = 1$. Moreover, $0 < \epsilon < \delta$ and the local 0 for process q is time ϵ for processes s and p. Then running the algorithm, in times relative to local time of q:
   - Time 0: Start of round 1 in q.
   - Time ϵ: s sends m to p then crashes.
   - Time δ: Start of round 2 in q.
   - Time $\delta + \epsilon$: m arrives to p. p sends m to q and delivers m.
   - Time 2δ: End of round 2 in q. q delivers SF, which violates Agreement.
   - Time $2\delta + \epsilon$: m arrives to q.

   (b) We need to ensure that the length of each round is atleast $\delta + \epsilon$ so that a message sent in round i is received by the end of that round in any processes local time. If that is not possible (rounds should be of length δ) instead of $t + 1$, we replace $2(t+1) + \lfloor \frac{\epsilon}{\delta} \rfloor$, as every sent message may take 2δ time to be processed by the recipient. Moreover, if $\epsilon >= \delta$, a different processes differ by a few rounds.

2. (a) Consider a run, which sender S sends message m to p and then crashes. Process p then delivers m, and crashes. So every correct process delivers SF, which violates Uniform Agreement.

   (b) Every process $p \neq s$ in round i does deliver m just after sending it to all and before halting. Moreover, before delivering SF in the final line, if it has received m in round $t + 1$, delivers m, otherwise delivers SF, and finally halts.

   Termination can be checked easily, as before every halt, a message is delivered.

   Validity is easy too, as the first line of algorithm says that sender sends m to all and then delivers m. So if it is correct, it would deliver m.

   Integrity is proofed by induction on the number of the round that the process receives message m. If we receive a message m in round i from a process q, the integrity of message m in round $i - 1$ for process q, proves the integirity of m in round i. As the base for the induction, the message in round 1 sent out by the sender satisfies the integrity property.

   For Uniform Agreement, first, if a process delivers SF, no correct process would deliver anything other than SF, as otherwise one such that correct process have received the message m before round $t + 1$ (as there are at most t faults), and so sent m to everyone before round $t + 1$, so no process can deliver SF in round $t + 1$. Now if a process p delivers $m \neq$SF, again, there should be some correct process (can be p itself, or s) in the path that m is transfered from s to p, that has received m before $t + 1$ (as there are at most t faults). So this correct process has sent m to everyone, so every correct process would deliver m in round $t + 1$.

3. (a) Algorithm:

In phase 1:

Sender s in round 1:

send m to all. go to phase 2.

Every receiver $p \neq s$ in round $1 \leq\ i \leq\ t+1$:

do

  if received some message m in round $i-1$

    then send m to all and go to phase 2.

  [receive round i messages]

od

Assume SF as received message and go to phase 2.

In phase 2:

Every process in round $t+2$:

send received message to all.

[receive round $t+2$ messages]

deliver the message received as more than half of the messages.

To proof the algorithm, we observe that as the phase 1 is pretty much the same as the original algorithm, Integrity, Validity, Agreement, and Termination of the original algorithm, now mean:

i. If a correct process sends a message m in round $t+2$ other than SF, then sender should have broadcasted m.
ii. If the sender is correct and broadcasted m, it sends it out in round $t+2$ too.
iii. If a correct process sends out m in round $t+2$, all correct ones send m too.
iv. Every correct process reaches phase 2.

Now, for the Integrity of the algorithm, as every one delivers the majority message m, at least some correct process should have sent out m in round $t+2$, and from fact i above, sender should have broadcasted m. By facts ii and iii above, if the sender is correct, then all correct processes send m in round $t+2$, and as $n > 2t$, then m should be delivered by sender too. For the Uniform Agreement, if m is delivered by any process, it should have been the majority, and so all correct processes get the same majority and deliver m. Termination follows from fact iv above.

(b) Assume an algorithm A exists that solves the problem. Divide the processes into two groups L and R, with sizes less than t. Let s be in L. Now run A, and let processes in L all crash from the round 1, so that no message gets to processes in R. Obviously by Termination and Integrity everyone in R should deliver SF. Now consider another run, which processes in R are faulty so that do not receive messages from processes in L. As they do not know that they are faulty, the scenario is the same as the first run, so all processes in R still deliver SF. By Uniform Agreement then everyone in L should deliver SF too. But it means that the sender s should deliver SF, which violates the Integrity. So no such algorithm A exists.

4. (a) These are just the differences from the original proof (available on the course page):

   The Termination argument is not the same, as we want to show that they terminate by round $r+2$. So assume that the last failure occures in round r, then everyone would decide by round $r+2$ and halt by $r+3$, as the *quiet* set of no process would change in round $r+2$, as there is no failures in round $r+1$.

   The proof of the claim in the proof changes too. The claim is that no process delivers SF before or in round i. It can be proved with the new algorithm too, as for each $1 \leq k \leq i$, process p_k crashes in round k, so the *quiet* set of all processes that have not halted grows in each round 1 to i, so they cannot deliver SF before or in round i.

   The rest of the proof is valid and need no change.

   (b) Consider 4 processes s, p, q, and r, which s is a faulty sender, and p is a faulty receiver, and $t = 2$. Now s simply sends m to p and crashes. p in turn in the first run, sends ? to q, but fails to send to r. In the second run, p sends m to q, and again fails to send it to r. Now r would deliver SF and halt, while q delivers m and halts, which violates Agreement.

5. (a) Lets a, b, c, and d be four processes. a starts sending messages to c , while b sens messages to d, and no other messages are sent. So the messages that both of them send get Lamport's Clock values $0, 1, 2, \ldots$, but for no message sent by a, and another one sent by b we can determine which one *happens before*, but well, we have lots of $C(a_i) < C(b_j)$ relations, that niether of them leads to $a_i \to b_j$.

   Using the definition of Lamport's Clock, and the observation above, given $C(a_i)$ and $C(b_j)$, all we can deduce is:

   - If $C(a_i) < C(b_j)$ then $b \not\to a$.
   - If $C(a_i) > C(b_j)$ then $a \not\to b$.
   - If $C(a_i) = C(b_j)$ then a and b are concurrent.

   (b) Each *timestamp* is a vector $(t_1, t_2, \ldots, t_n)$ which holds for each process a single time t_i. Each process maintains a current timestamp also. Moreover, $a_i \to b_j$ if and only if $\forall\ 1 \leq k \leq n : C(a_i)_k <= C(b_j)_k$.

   - Initially: Each process sets its timestime to a vector of zeros.
   - Process k on sending a message: The process first increases C_k by one, then attaches its current timestamp C to the message.
   - Process p on receiving a message a_i, sets its current timestamp to the max vector of its current timestamp and the one from received message. Means $\forall\ 1 \leq k \leq n : C_k := \max(C_k, C(a_i)_k)$.