The University of Texas
at Austin

CS371D Distributed System

Problem Set 02

Edited by LaTeX

Department of Computer Science

STUDENT
Jimmy Lin
xl5224

INSTRUCTOR
Lorenzo Alvisi

TASSISTANT
Chao Xie

RELEASE DATE
March. 19 2014

DUE DATE
March. 26 2014

TIME SPENT
20 hours

March 26, 2014

Contents

1 Problem 1: Early-Stopping Algorithm

According to the class note, we first list out the property required by one protocol if it is a correct t-tolerant early-stopping protocol as follows:
Validity: If the sender is correct and broadcasts a message m, then all correct processes eventually deliver m.
Integrity: Every correct process delivers at most one message, and if it delivers $m \neq SF$, then some process must have broadcast m.
Agreement: If a correct process delivers a message m, then all correct processes eventually deliver m.
Termination: Every correct process eventually delivers some message.

1.1 Under Crash Failures

Proof for Validity. A message m sent from a correct sender to all processes at round 1 will be received by every correct process at the end of round 1. This is true because of the validity of underlying *send* and *receive*. Hence, in this protocol, every correct process will deliver m by the end of round 1. □

Proof for Integrity. The new protocol shows that each process delivers *atmost* one message and then halts. The delivered message could either be SF or m depending upon the sender action in round 1. For a process p to deliver m in a round $r \neq 1$, it must have received the value from some other process q. In this form, we can trace back the chain whose terminal point is the sender s that broadcast m. (Lemma 1 of the class note.) □

Proof for Agreement. There can be no round where a process set value to SF and another process set value to m. This is proved as follows. Consider a round k when process p decides m and process q decides SF. For process p to decide m it either heard m from the sender in round 1 or m was relayed to it by some other process. For q to decide SF, it must have gone through two consecutive rounds (say i, j) hearing only question mark $'?'$. This is a contradiction as, if p had heard the value m in round i it would have broadcast m in round j. Since all channels are reliable, q is will hear m in round j for sure. Hence, we can conclude that no two processes can in the same round deliver different messages.

□

Proof for Termination. Based on the statement at line 6 and 8, if in any round a process receives a "non-?" value, then it delivers the value in that round. Based on the new statemnt at line 9 (actually it is line 10), if ? value is received, either it is in $f + 1$ round or there are no new failure observed current round, SF (sender fails) is delivered. Hence, we can conclude from the analysis of two cases above that Termination property of new protocol holds. □

1.2 Under Send-Omission Failures

We can disprove the correctness of the above protocol under existence of send omission by . Postulate that a process p exhibits send omission when it omits to send a message m to some or all of processes. Consider that in round 1, s omits to send message m to some processes, say, P. Hence, we have $\forall p \in P, |faulty(p, 1)| = 1$. Then in Round 2, the sender now sends m to a subset of the processes P and their are no other faults. Those process receiving m will deliver m. But there are some processes, say P_1, still not get m will set their $\forall p' \in P_1, |faulty(p, 2)| = 1$. For the set of processes not receiving m, no new faults have occurred, which satisfies $|faulty(p, k)| = |faulty(p, k - 1)|$ (for here, $k = 2$). Therefore, they will decide that the sender is faulty, deliver SF and halt. For now, we have some correct processes deliver m and some correct deliver SF. But it is apparent that this violates the property of agreement.

2 Problem 2: Terminating Reliable Broadst

2.1 Algorithmic Description

In order to solve TRB in $f + 2$ rounds, we first specify the behavior of sender and receiver for round 1 and then reference to the existing $f + 1$ consensus algorithm to decide (deliver) m.

```
/*##########################*\
#  Sender s in first round  #
\*##########################*/
  send m to all other processes

/*##########################*\
#  Receiver in first round  #
\*##########################*/
  receive m from s (it can only be m)
  if m is received,
      then val = m
  else val = SF .
```

```
// For this part, we directly reference the consensus algorithm in class note

// For the next f+1 rounds each process i executes the following
Initially V = {v_i}  // value received from the first round
To execute propose(v_i):
    round k, 1<=k<=f+1
  send v_i to all
  for all j, 0<=j<=n-1, j != i  do
      receive value from process p_j
      V := V U value

// Process decide the consensus value as
To execute decide(x):
  if k = f+1:
      decide V
```

2.2 Correctness Check

Proof of Termination. It is obvious that no matter what sender broadcast at the first round, the new algorithm provided will send eventually deliver message m or SF, in terms of the termination property of consensus algorithm. □

Proof of Validity. Since we only postulate the existence of crash failure, a correct sender broadcasting message m will not be omitted. Hence, according to the underlying *send* and *receive*, the message m broadcast by the sender at round 1 will be obtained by all correct process. For round 2 to round $f+1$, due to the validity condition of consensus algorithm, all correct processes will eventually decide on m and therefore delivering m. □

Proof of Agreement. If a process deliver a message m, it must be value derived by the consensus algorithm (consensus value). By the agreement property of the consensus algorithm, we can say that m will be also delivered by all correct processes. That is to say, a process can only deliver a consensus value as all the other processes in the system do. □

Proof of Integrity. Obviously, in the consensus algorithm, each process will eventually decide one value. Thus, no matter what value is initialized, we can say that every correct process will consequently deliver *atmost* one message for the resulted TRB algorithm. And following the integrity property of the consensus algorithm, a message m delivered by one process must be consensus message. And this consensus message must be the one broadcast by the sender in round 1 (the *send* and *receive* are guaranteed). For a process to deliver m which is also the consensus message it has to be proposed by some process in the system. This follows from the integrity property of the consensus algorithm. By the algorithm given this is the m that would have been broadcast by the sender in round 1. The algorithm maintains the channel *send* and *receive* property so message m will be received by all correct process at the end of round 1. From there it will be proposed and eventually delivered after $f+1$ rounds. □

3 Problem 3: Uniform TRB

3.1 Round-based Algorithm

3.2 Unsolvable if $n \leq 2t$

Proof by Contradiction. Assume that there is a protocol that solves the uniform TRB under the condition of faulty processes $t \geq n/2$. And then let us follow the hint to partition the processes in the system into two disjoint sets S_0 and S_1, such that S_0 includes t processes (sender inclusive), and S_1 includes the other $n - t$ processes. And we specify processes in S_0 be correct and all processes in set S_1 to be faulty. Note that by general omission failure, it can be send omission and receive omission. That is, process in S_1 do neither receive messages (receive omission) nor send messages (send omission) to processes in set S_0. Now we start the reasoning. When the sender in S_0 sends a message m, say at round r, all processes in S_0 receive the message and deliver it. However, processes in S_1 do not receive message because of receive omission. In the next round $r + 1$, all processes will broadcast their received value to other processes. Nevertheless, processes in S_1 send to each other $'?'$ since they have receive nothing in round r. But Processes in S_0 send m to all processes since they are correct processes and thus receive value from last round r. By the same reasoning, processes in S_1 will not receive a value and eventualy deliver SF when the protocol ends. But processes in S_0 deliver the first valid message m. Hence, we get a situation where majority processes deliver SF and minority processes deliver initial message m, which violates uniform agreement. And even we turn to majority voting to find the correct message, since the majority processes are faulty (not communicable), we cannot eventually decide the real value (recover) and reach uniform agreement. Therefore there exists no algorithm to achieve uniform agreement if $2t \geq n$.

□