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 **2**
1.1 Under Crash Failures 2
1.2 Under Send-Omission Failures 2

2 Problem 2: Terminating Reliable Broadst **3**
2.1 Algorithmic Description 3
2.2 Correctness Check 4

3 Problem 3: Uniform TRB **5**
3.1 5

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. Obviously, 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}
To execute propose(v_i):
    round k, 1<=k<=f+1
  send {v in V: p_i has not already sent } to all
  for all j, 0<=j<=n-1, j != i  do
      receive S_j from process p_j
      V := V U S_j

// 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.

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 $Round1$. The algorithm maintains the channel *send* and *receive* property so message m will be received by all correct process at the end of $Round1$. From there it will be proposed and eventually delivered after $f+1$ rounds. □

3 Problem 3: Uniform TRB

3.1

The Uniform Agreement property for a Terminating Reliable Broadcast states that, If any process (whether correct or faulty) delivers a message m, then all correct processes eventually deliver m.

To achieve uniform TRB we delay delivering of messages right until the end. So, after $t+1$ rounds of message exchange the protocol goes through an additional round to ascertain the majority of the messages about to be delivered in the system. The protocol works on the assumption that majority (maybe simple) of the processes are not faulty. So if a correct process receives $j > n/2$ $m's$ it (and other alive correct processes) can safely deliver m. Otherwise a SF is delivered. Delaying the message delivery after accounting for the t crashes allows processes to deliver the same value as decided by majority correct processes.

```
1:  Every process p executes the following algorithm:
2:  if process = sender then val = m else val =?
3:  for round k, 1 <= k <= t + 1
4:      Send val to all
5:      if val = m then break;
6:      receive round k messages;
7:      if received some message m != '?' then val = m
```

By now we have all processes ready to deliver some m or ready to decide SF. In order to ensure that all processes (correct and faulty) decide the same we need another round for the processes to exchange their intentions.

```
1:  Send val to all processes;
2:  Receive round t + 2 values from all.
3:  if received majority (atleast simple) values of message m
4:      deliver m
5:  else
6:  if received majority (atleast simple)  values as ?
7:      then deliver SF
8:  halt
```

Proving the algorithm

Proof for uniform TRB

Validity: If the sender is correct and broadcasts a message m, then all correct processes eventually deliver m.

When a correct sender broadcasts m in round 1 it will be heard by all the correct processes. This follows from the validity of the underlying channel. The processes hearing this value will now be part of the majority required for all processes to decide m. As, the protocol assumes that a majority of the processes are not faulty we will have the correct processes sending m in round $f+2$ leading to all correct alive processes in the system to deliver m.

Uniform agreement: If any process (correct or faulty) delivers a message m, then all correct processes eventually deliver m

For uniform agreement to fail some processes have to deliver m and others have to deliver SF. This cannot happen in the algorithm suggested.

Case 1 - p, a non-faulty process, delivers m. For process p to deliver m it has to hear a majority of m messages in round $t+2$. Simultaneously a correct process q heard a majority of '?'. This means that one process sent '?' to q and m to p. As there are no byzantine faults we can conclude that the once a correct process delivers m all correct processes will deliver m.

Case 1 - p, a faulty process, delivers m. For a faulty process p to deliver m it has to hear a majority of m messages in round $t+2$. The process could be behaving faulty in previous rounds but when it decides to deliver m there must have been a majority of m that it obtained. So all correct processes, who would have also heard the *same* majority, will deliver m.

Integrity: Every correct process delivers at most one message, and if it decides $m! = SF$, then some process must have broadcast m.

Each correct process delivers *atmost* one message, either m or SF based on the majority it sees. For a correct process to deliver $m! = SF$ at the end of $t+2$ rounds it must have heard a majority of m messages in that round. For a majority of m to exist after round $t+2$, more than half of the processes must have m with them in round $t+1$. For them to have m, either they obtained it directly from the sender or via a chain of other processes. In either case m had to be broadcast in round 1 for processes to have it by round $f+1$.

Termination: Every correct process eventually delivers some message

At the end of round $t+2$ all the alive correct process will reach a common decision and deliver either m or SF based on the majority of the values heard. This is possible due to the fact that reliable channels ensure that m sent in $t+2$ round will be received in the same round. There could be processes though that do not terminate after $f+2$ rounds as they exhibit general omission and thus, are faulty. As the termination condition governs for only correct processes it is not mandatory for faulty processes to terminate.

No Uniform TRB where n $\leq$ 2t

Following is the proof that there is no protocol solving the Uniform TRB problem in a synchronous system where up to $t \geq n/2$ processes can commit omission failures.

Proof by contradiction: Let us assume that there is a protocol that solves the uniform TRB in presence of up to $t \geq n/2$ faulty processes.

Let us partition the processes in the system in two disjoint sets P_0 and P_1 such that P_0 includes t processes (including the sender), and P_1 includes the other $n-t$ processes. Consider the following execution:
Let processes in P_0 be correct and all processes in set P_1 be faulty. Processes in P_1 send and receive all messages within the set P_1. They however do neither receive (receive omission) nor send messages (send omission) with respect to processes in set P_0. When the sender in P_0 sends a message m, all in P_0 receive the message. Processes in P_1 oblivious to the send event of m (receive omission) obtain nothing. In the next round all processes broadcast their received value. Processes in P_0 send m to all processes. Processes in P_1 send to each other $'?'$ as they exhibit receive omission from processes in P_0. When the protocol ends, processes in P_1 have still not heard a value and consequently deliver SF. Processes in P_0 on the other hand deliver the first valid message, ie m. This violates uniform agreement because some faulty processes delivered SF and the correct processes delivered m. Hence there exists no algorithm.