Gaurav Nanda **Problem Set #2**
Distributed Computing **October 19, 2013**

Solution 1-A: **Prove that it is t-tolerant early stopping TRB for crash failure.**

Termination: Every correct process eventually delivers some message.

If in any round a process receives a value, then it delivers the value in that round. If a process has received only ? for f + 1 rounds, then it delivers SF in round f+1. Therefore, at the end of f+1 rounds, every process has delivered some message.

Validity: If the sender is correct and broadcasts a message, then all correct processes eventaully delivers *m*.

If the sender is correct, then it sends m to all in round 1. By validity of underlying send and receive, every correct process would receive message *m* by the end of round 1. By the protocol, every correct process would deliver *m* by the end of round 1.

Integrity: Every correct process delivers atmost one message, and if it delivers m $\neq$ SF, then some process must have broadcast m.

1. Only crash failures can occur and once a process delivers an event, it halts in the next round. Therefore a process cannot deliver more than one message.

Lemma 1

For any r $\geq$ 1, if a process p delivers $m \neq SF$ in round r, then there must exists a sequence of processes p_0, p_1,..., p_r, such that p_0 = sender and p_r = p, and in each round k, 1 $\leq$ k $\leq$ r, p_{k-1} sends *m* and p_k received it. Furthermore, all processes in the sequence are distinct, unless r = 1 and $p_0 = p_1$ = sender.

2. From *Lemma* 1, it follows that if a process has delivered message *m* in some round, then there exists a chain starting from the sender to the process which is delivering the message. Therefore, we can assert that there exists a process which broadcasted the message.

Agreement: If a correct process delivers a message *m*, then all correct processes eventually deliver *m*.

If a process p set its value to *m*, then in the next round all correct processes may (depending upon whether p was correct or faulty) deliver *m*. Similarly, if a process set its value to *SF*, then in the next round all the correct processes may deliver *SF*. Therefore, we should prove no two processes can set their value to *m* and *SF* in the same round.

Lemma 2

It is impossible for p and q, not necessarily correct, to set value in the same round r to m and SF, respectively.

Lemma 2 proof:

Let us assume two different process p_r and q_r set their value to *m* and *SF* in the round r respectively.

If p has received *m* in round r, then there exists a distinct p_{r-1}, which sent *m* to process p_r in the round r-1.

If p_{r-1} was correct, then ideally q_r should also have received m. Therefore, it could not have delivered SF. Otherwse, if p_{r-1} was not correct and **only crash faliures are allowed**, then q_r should have added it to its faulty set in the round r. Therefore, $|\text{faulty}(p, k)| \neq |\text{faulty}(p, k-1)|$.

In both scenarios, there cannot exist any q_r which delivers SF. Hence, the CONTRADICTION !!

Agreement-Proof

1) If no correct process ever receives m, then every correct process delivers SF in round f +1.

2) Let r be the earliest round in which a correct process delivers value $\neq$ SF,

If $r \leq f$:

* By Lemma 2, no (correct) process can set value differently in round r.

* In round r+1 $\leq f$ +1, that correct process sends its value to all.

* Every correct process receives and delivers the value in round r+1 $\leq f$ +1.

If $r = f + 1$:

* By Lemma 1, there exists a sequence p_0,...,p_{f+1} = p_r of distinct processes

* Consider processes p_0, ..., p_f

  - f+1 processes; only f faulty.
  - one of p_0, ..., p_f is correct– let it be p_c
  - To send m in round c+1, p_c must have set its value to m and delivered m in round c $\leq$ r.

CONTRADICTION !!

<u>Solution 1-B</u>: **Prove that it is not correct t-tolerant early stopping TRB for send-omission failure.**

If we are able to prove that Agreement does not hold, we are done. To show that agreement does not hold, we should prove that two processes are able to deliver m and SF in the same round.

Let us imagine there are 4 process, p_0, p_1, p_2, p_3.

Round-1

As the send-omissions are allowed, p_0 sends m only to p_1 and p_1 does not send ? to p_3.

By the end of round-1, $|\text{faulty}(p_2)| = 1$ and $|\text{faulty}(p_3)| = 2$.

Round-2

p_1 sends m only to p_2 and p_2 delivers m. However, p_3 does not receive m, neither is its faulty count increased, therefore it thinks that $|\text{faulty}(p, k)| = |\text{faulty}(p, k-1)|$ and delivers SF.

This violates the agreement property. Hence, we assert that it is not correct t-tolerant early stopping TRB for send-omission failure.

Solution 2: **Write an algorithm for TRB using an algorithm for consensus.**

Algorithm 1 TRB using Consensus

```
**Round-1**
1. Sender sends message $m$ to all.
2. Sender delivers the message and halts.
3. Rest of the processes receive the message sent by the sender. If they received a message $m$
from the sender, they propose $m$, else $SF$ in the consensus algorithm.
**Following rounds**
1. Rest of the processes run the consensus algorithm for $f + 1$ rounds.
2. At the end of consensus protocol, all the process deliver their decided value.
```

Termination: Every correct process eventually delivers some message.

If the sender is correct, it would deliver the message m at the end of the round 1 itself. Using the termination condition of the consensus protocol (Every correct process eventually decides some value), other correct processes would deliver either m or SF at the end of consensus protocol, i.e. f + 2 rounds.

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

If the sender is correct, sender would deliver m at the end of the round one. By validity of the underlying send and receive, every correct process will receive m by the end of round 1.

Once all the processes propose m, by the validity of consensus protocol with the crash failure assumption (i.e. If all processes that propose a value propose v , then all correct processes eventually decide v), all the processes would decide m and deliver m at the end of f+2 rounds.

Integrity: Every correct process delivers atmost one message, and if it delivers m $\neq$ SF, then some process must have broadcast m.

1. Correct sender delivers only one message at the end of round 1 (no byzartine failures) and then it halts. Any other correct process, by the integrity of the consensus protocol (Every correct process decides at most one value, and if it decides v, then some process must have proposed v), would decide at most one value and would deliver it after the consensus protocol terminates, i.e. after f + 2 rounds.

2. Every other correct process(non-sender) delivers at the end of the round f+2. If they deliver $m \neq SF$, it must have been proposed at the beginning of the consensus protocol by some process (Integrity property of consensus protocol). Only possible way a process can propose m in the consensus protocol is if it receives that message from the sender in the round - 1. Therefore, if a process deliver message m, it must have been broadcasted by some process.

Agreement: If a correct process delivers a message m, then all correct processes eventually deliver m.

1) If the sender delivers m, it means it was successful in sending m to all the correct processes (Because only crash failures are allowed). Once all the correct processes receive m, they would propose m in the consensus algorithm. Following the validity of consensus (If all processes that propose a value propose v , then all correct processes eventually decide v.), all correct processes

would decide upon m and would then deliver m at the end of round $f + 2$.

2) If the sender was faulty, it could have sent m to few processes and died. Processes which received m would propose it and other would propose SF. Leveraging the agreement of the consensus (If one correct process decides v, all correct processes would eventually decide v), all processes would decide the same value, either m or SF and then deliver it.

Therefore, agreement would always hold.

<u>Solution 3-A</u>: **Write an algorithm for Uniform-TRB.**

Algorithm 2 Uniform TRB assuming f faulty process. Total process > 2f.

```
Initialize:
for p_i, 1 ≤ i ≤ n do
    if $p_i = sender$ then
        $p_i(value) \leftarrow m$
    else
        $p_i(value) \leftarrow$ "?"
    end if
    $p_i(sentToAllFlag) \leftarrow False$
end for

Processes in the following rounds:
for round k, 1 ≤ k ≤ f + 1 do
    if $process(value) \neq$ "?" AND $process(sentToAllFlag) = False$ then
        send m to all.
        $process(sentToAllFlag) \leftarrow True$
    end if
    Receive round k values from all.
    if received value $v \neq$ "?" then
        $process(value) \leftarrow v$
    end if
end for

Final Round:
Every process sends its value to all.
Every process receives response from all.
if process receives atleast f+1(majority) values = $m$ then
    Deliver $m$
elseprocess receives atleast f+1(majority) values = ?
    Deliver $SF$
end if
Halt
```

Termination: Every correct process eventually delivers some message.

At the end of the f+2(final) round, every correct process would either deliver SF or m after consulting with other processes.

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

If the sender is correct, then it sends m to all in round 1. By validity of underlying send and receive, every correct process would receive message m by the end of round 1. By the protocol, every correct process would retain its value to be m till f+1 rounds.

In the final round, as there are atleast f+1 correct processes, all the correct processes would receive

m from a majority of processes. Therefore, all correct process would deliver m by the end of final round.

Integrity: Every correct process delivers atmost one message, and if it delivers m $\neq$ SF, then some process must have broadcast m.

1) Messages are delivered only in the final round, also the byzartine failures are not allowed. Therefore, every correct process delivers atmost one message.

2) *Lemma* 1

For any r $\geq$ 1, if a process p sets it value to $m \neq SF$ in round r, then there must exists a sequence of processes p_0, p_1,..., p_r, such that p_0 = sender and p_r = p, and in each round k, 1 $\leq$ k $\leq$ r, p_{k-1} sends m and P_k received it. Furthermore, all processes in the sequence are distinct, unless r = 1 and p_0 = p_1 = sender.

A process delivers a message $m \neq SF$, if and only if it receives m from a majority of processes in the last round $(f + 2)$. From *Lemma* 1, it follows that if a process has set its value to m in some round, then that exists a chain starting from the sender to the process which has set its value to m. As the byzartine failures are not allowed and every process in the majority set (having m), should have got m from the sender, we can assert that there exists a process which broadcasted the message.

Uniform Agreement: If a process (correct or faulty) delivers a message m, then all correct processes eventually delivers m.

A) CASE: Process delivers $m \neq SF$.

A process can deliver m, if only if it gets m from a majority of the processes in the round f+2. Let us assume that one process received majority of m in the final round. This ensures that atleast one of the correct process would have set its value to m. If we can prove that, if one correct process has set its value to m before the start of last round, other correct processes should have also set their value to m before the start of last round, we would be certain that all the correct processes would deliver m in the last round.

1) If the sender was correct, all the correct processes should have received the message m in the round one itself and would have retained it till round f+2.

2) If the sender was faulty and if the first correct process receives m at the end of the round r, and then it should deliver m to all the correct processes in the round r + 1. Therefore, we need to prove that f+1 cannot be the earliest round r in which a correct process receives m. Because if that is the case, at the start of round f + 2, only one correct process would have m.

Let r be the earliest round in which a process p_r sets its value to m.

If $r \leq f$:

* In round r+1 $\leq f$ +1, that correct process sends its value to all.
* Every correct process receives and sets its the value in round r+1 $\leq f$ +1.

If $r = f + 1$:

* By Lemma 1, there exists a sequence p_0,...,p_{f+1} = p_r of distinct processes

* Consider processes p_0, ..., p_f

- f+1 processes; only f faulty.
- one of p_0, ..., p_f is correct– let it be p_c.
- To send m in round c+1, p_c must have set its value to m in round c $\leq$ r.

CONTRADICTION!!

Therefore, the first correct process to receive m must have received it in round $r <$ f +1 and would have delivered it to all the correct processes in the round f + 1.

This would ensure that majority of the process having m in the final round would include all the correct processes. Hence the Uniform Agreement would prevail.

B) CASE: Process delivers $m = SF$.

A process can deliver SF, if and only if, it gets ? from a majority of the processes in the round f+2. If the process has received ? from a majority of processes, we are certain that atleast one of the correct process has its value set to ?.

As we discussed in the previous section, if a correct process has set its value to $m \neq ?$, before the start of the last round, all the other correct processes should have also set their value to m by the last round. Therefore, when we see that one correct process has its value set to ?, we are certain that none of the correct process has set its value to $m \neq ?$.

Therefore, any process receiving majority of ? can deliver SF, because that is what each of the correct process would also do, after receiving a majority of ? in the same round. Hence the Uniform Agreement would prevail.

Solution 4: **Does above protocol solve consensus?**

No. This algorithm does not satisfy the Integrity property of the consensus protocol.

Integrity suggests that ”Every correct process decides at most one value, and if it decides v, then some process must have proposed v.”

We can show with the help of an example that integrity does not hold in this algorithm, i.e. a process can end up deciding some value v which was not proposed by any process.

Example:

Let us assume that there are 3 processes participating in the consensus protocol, each proposing 4 (100), 5(101) and 6(110) respectively.

* For bit-1, (1, 1, 1) are the proposed values, following the validity, they would also decide 1. $d_1 = 1$.

* For bit-2, (0, 0, 1) are the proposed values, consesus can decide upon either 0 or 1. Let us assume they decide 1. $d_2 = 1$.

* Similarly for bit-3, (0, 1, 0) are the proposed values, consesus can decide upon either 0 or 1. Let us assume they decide 1. $d_3 = 1$

That is they decide 111 (7), which was not a proposed value. Hence, the algorithm does not solve consensus and violates the integrity property of the consensus.