HW3 Solutions: CS425 FA18

1. (Solution and Grading by: <Le>)

Counter example showing the algorithm would fail:

Assuming there are N=6 processes, N>5: Each process propose their id as vote (vi), vi being the minimum vote.

Round 1:

P1 sends v1 to P2, and P1 fails. P2 has v1 and the others don’t.

Round 2:

P2 sends v1 to P3, and P2 fails. P3 has v1 and others don’t.

Round 3:

No messages are sent. (P4 fails)

Round 4:

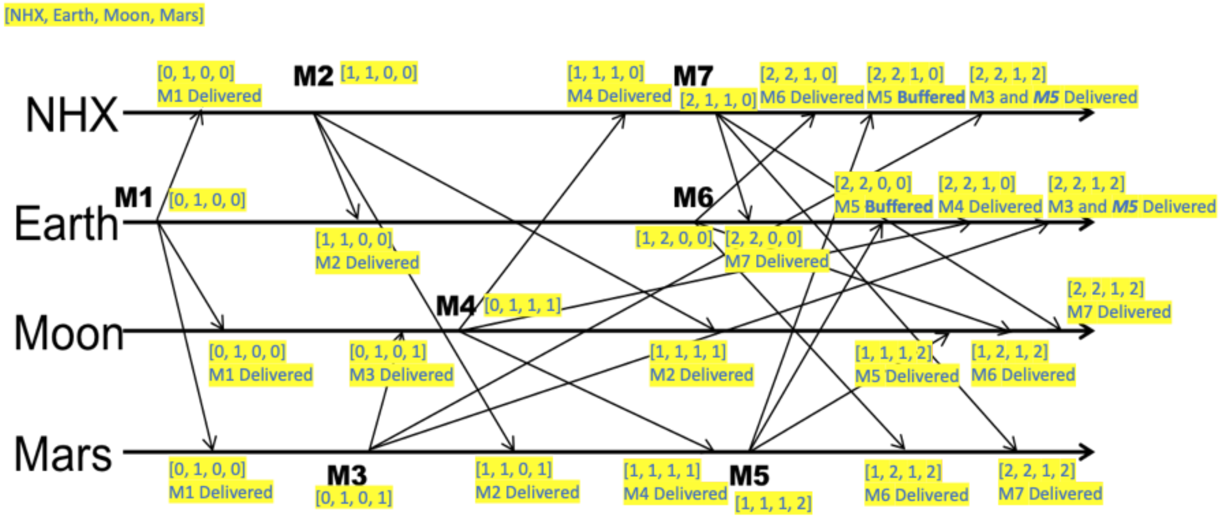
No messages are sent (no new messages from last round). (P5 fails)

Round 5:

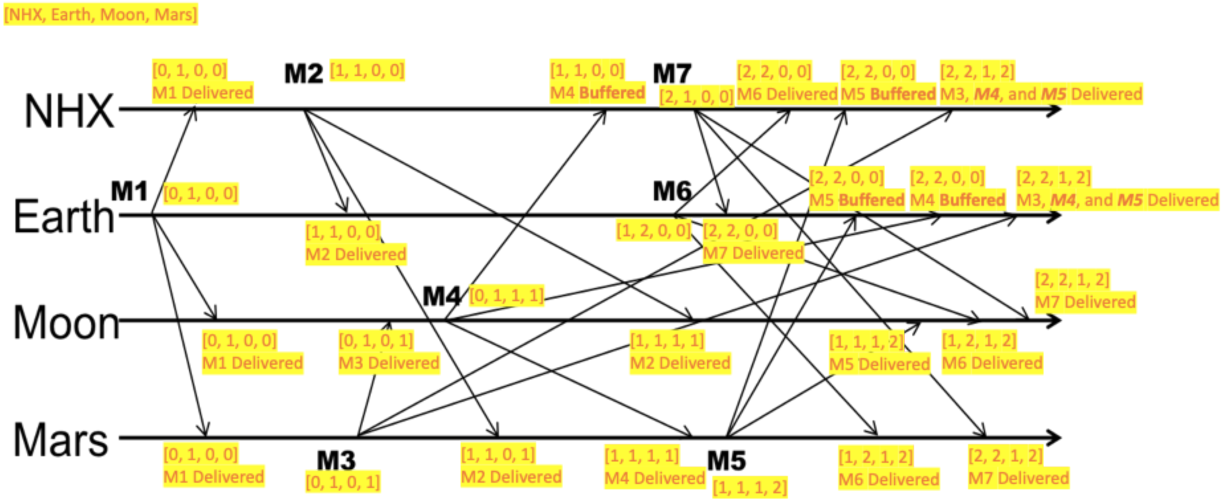
No messages are sent since there are no new messages received from last round

So far there have been 5 rounds and 4 failures however P3 is the only process that holds the global minimum value (v1) - process will converge to different value. So the algorithm does not achieve consensus.

2. (Solution and Grading by: <TA, Beomyeol>)


[NHX, Earth, Moon, Mars]
NHX
Earth
Moon
Mars
M1
M2
M3
M4
M5
M6
M7
[0, 1, 0, 0] M1 Delivered
[1, 1, 0, 0]
[1, 1, 1, 0] M4 Delivered
[2, 1, 1, 0]
[2, 2, 1, 0] M6 Delivered
[2, 2, 1, 0] M5 Buffered
[2, 2, 1, 2] M3 and M5 Delivered
[0, 1, 0, 0]
[1, 1, 0, 0] M2 Delivered
[1, 2, 0, 0]
[2, 2, 0, 0] M7 Delivered
[2, 2, 0, 0] M5 Buffered
[2, 2, 1, 0] M4 Delivered
[2, 2, 1, 2] M3 and M5 Delivered
[0, 1, 0, 0] M1 Delivered
[0, 1, 0, 1] M3 Delivered
[0, 1, 1, 1]
[1, 1, 1, 1] M2 Delivered
[1, 1, 1, 2] M5 Delivered
[1, 2, 1, 2] M6 Delivered
[2, 2, 1, 2] M7 Delivered
[0, 1, 0, 0] M1 Delivered
[0, 1, 0, 1]
[1, 1, 0, 1] M2 Delivered
[1, 1, 1, 1] M4 Delivered
[1, 1, 1, 2]
[1, 2, 1, 2] M6 Delivered
[2, 2, 1, 2] M7 Delivered

3. (Solution and Grading by: <TA, Beomyeol>)



4. (Solution and Grading by: <Zhuolun>)

   Possible order 1: M11, M12, M21, M31, M32, M33, M22, M23, M13

   Possible order 2: M11, M12, M21, M31, M32, M33, M22, M13, M23

5. (Solution and Grading by: <Rui>)

   Solution:

   A review for concepts:

   Safety: If some round has L/quorum(depends on the number in each sub-question) hearing proposed value v and accepting it (middle of Phase 2), then subsequently at each round either: 1) the round chooses v as decision or 2) the round fails.

   Eventual liveness: If things go well sometime in the future (messages, failures, etc.), there is a good chance consensus will be reached.

   i.

   a. Yes. When everything goes well, Election, Bill, and Law stage will all finish within timeout and go into the next round. If two leaders are elected and one of them don't get quorum of OK in round r, it will timeout and start a new round. Because quorum of members already agree with a value v in round r, the isolated leader will get the decided value v quorum times in his round (r+1) and take it as decision. Thus, the consensus is reached.

   b. Yes. There might be two leaders in the network at the same time, getting 5N/12 +1 votes each. However, in Bill phase, there will be only one leader decide get quorum and go into law stage. Thus, all members will only get one result. The second leader will timeout and starts a new round.

c. Faster. It needs less time to elect a leader because it needs to wait for less OKs. (Also accept: Slower. It might take more time to reach consensus for all process because the 2nd leader might need one more round.)

ii.

a. Yes. When everything goes well, Election, Bill, and Law stage will all finish within timeout and go into the next round.

b. No. The decided value in the past round (quorum agree) might not be the decision value in the Bill phase. This violates the safety definition.

c. Faster. It needs less time to bill because it needs to wait for less OKs.

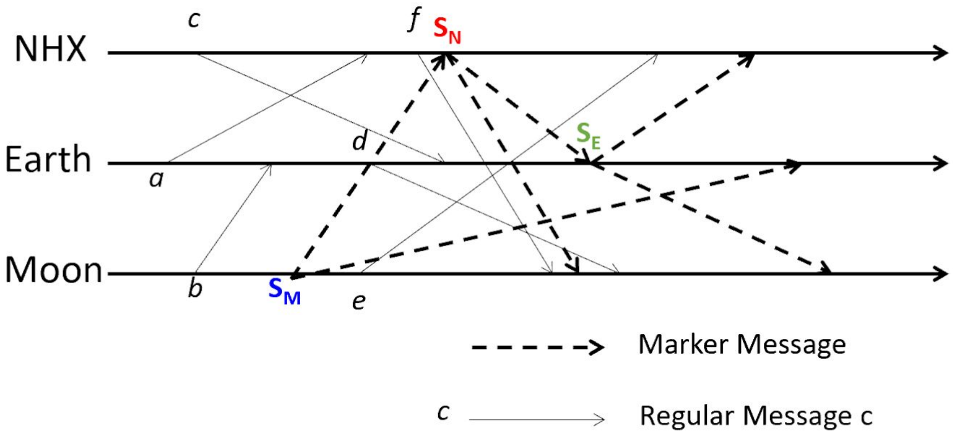
iii.

a. Yes. When everything goes well, Election, Bill, and Law stage will all finish within timeout and go into the next round.

b. No. There might be two leaders in the network at the same time, getting 5N/12 +1 votes each. If they decide different values in the bill stage, different processes might get different values in Law stage.

c. Faster. It needs less time to elect and bill because it needs to wait for less OKs for both stage. (Also accept: Slower. It might take more rounds to reach consensus.)

6. (Solution and Grading by: <Shegufta>)



NHX (N)	Earth (E)	Moon (M)
S_N	S_E	S_M
$C_{EN} = \{\}$	$C_{NE} = \{\}$	$C_{NM} = \{f\}$
$C_{MN} = \{\}$	$C_{ME} = \{\}$	$C_{EM} = \{d\}$

7. (Solution and Grading by: <Rahul>)

Assumptions:

- Every node has a complete membership list.
- Algorithm fails if the total number of nodes < k

Protocol:

1. Process on detecting failure among any of the k leaders sends **ELECTION** message to all the other nodes.
2. Node on receiving **ELECTION** message will send **OK** message to all processes with higher ids than it, and then wait a period of time to receive **OK** messages.
3. All processes with more than k **OK** messages will wait for **COORDINATOR** messages.
4. All processes with less than k **OK** messages will claim itself to be the leader and send **COORDINATOR** message to all processes.
5. Any process that did not receive k **COORDINATOR** messages will re-initiate the election.

Safety:

Safety is guaranteed as OK message is sent to all processes which have higher id. Only k nodes will receive less than k OK messages and they will claim themselves to be the leader.

Liveness:

In our protocol we define timeouts for receiving OK and coordinator messages, hence we can be sure that the protocol will terminate. In case any process received less than k COORDINATOR messages it will re-start the protocol. Thus we can say that the protocol satisfies liveness.

8. (Solution and Grading by: <Faria>)

a. There are a total of *k* token messages per file in the ring. If a process that wishes to read receives a token, it can go ahead and read the file. However, to write to a file, a process (P1) either has to has to acquire all *k* read tokens. The process retains all k tokens while entering the critical section.

Tie breaking methods: If two processes are competing to acquire all write tokens, then we use a random timeout on each of the processes to backoff. Then, the processes that would like to write can try acquiring all k tokens again.

b. **Safety**: Safety is guaranteed because the algorithm ensures the following:

I. If any of the tokens are being used for reading, no writes can progress as processes need to acquire all tokens first.

II. If the token is in the write state, no reads can progress as all k tokens are acquired by the writer.

III. As *k* tokens need to be acquired to write, at most 1 process can write at a time.

Liveness: Given no failures and lost messages, liveness is guaranteed as a process eventually gets to read/write from/to a file as tokens will be propagated around the ring once other processes are done with their work. Additionally, to break ties between processes who want to write at the same time, we can circulate the process IDs and deterministically give the greater process the chance to write first.

C.

Bandwidth: the total number of messages sent in each enter and exit operation. (We ignore the number of messages required to select the process that gets to write first and just focus on bandwidth required by circulating tokens).

Client delay: the delay incurred by a process at each enter and exit operation (when no other process is in, or waiting)

Synchronization delay: the time interval between one process exiting the critical section and the next process entering it (when there is only one process waiting)

Reads

	Enter	**Exit**
Bandwidth Usage	1 message by requesting process (also acceptable: upto N messages in the system, e.g.,if all tokens were held by successor for a write)	1 message
	Worst Case: The successor has all the tokens e.g., it was writing	
Client Delay (for the enter operation)	N message latencies	
Synchronization Delay	(N -1) message latencies	

Writes

	Enter	**Exit**
Bandwidth Usage	Best case: 1 message Worst case: *k* message by requesting process (also acceptable: upto k*N messages in the system, e.g.,if all tokens were held by successor for a write and then were converted to read tokens in the ring)	K messages (all tokens released together). Also acceptable -- 1 message holding all tokens.
	Worst Case: The successor has all the tokens e.g., it was writing	
Client Delay (for the enter operation)	N message latencies	
Synchronization Delay	(N -1) message latencies	

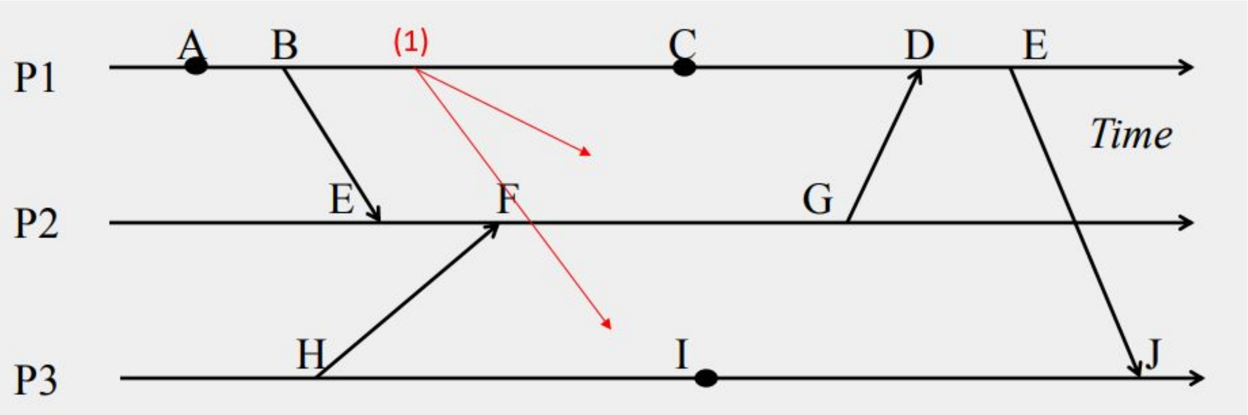
General comments:
Grading has been done very leniently for part c, as long as the student's solution makes sense.

However, many students have given answers for latency in the order of k*N. This is because students think that if k tokens need to be circulated around the ring, their individual latency needs to be added up -- this is not true. If k tokens circulate around the ring once, latency is still only N. No points have been deducted though.

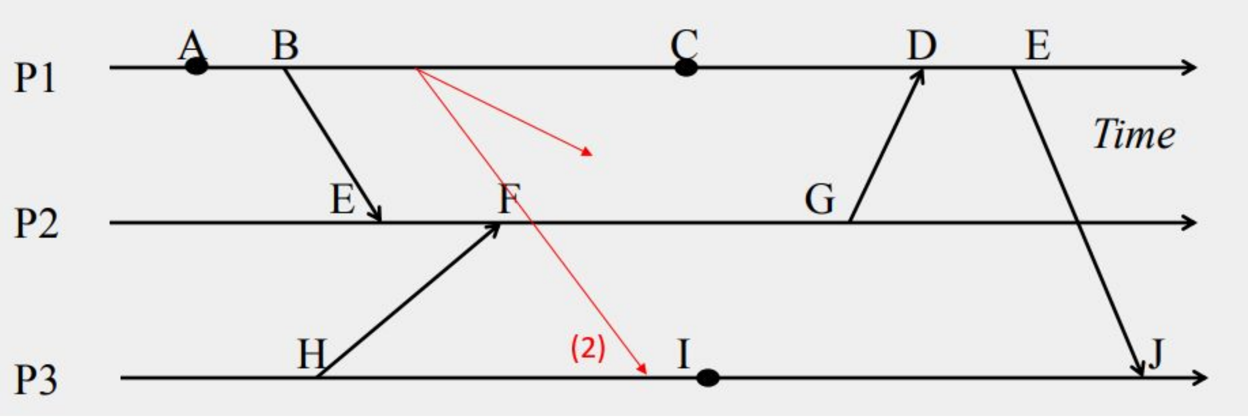
9. (Solution and Grading by: <Rui>)
   (The fix for incorrect virtual synchrony is flexible as long as it works.)
   a. Incorrect. M32 was sent in V11 but delivered in V12. Correction: p1 & p2 deliver M32 before V12.
   b. Correct. Every non-faulty process received the same set of multicasts and everything in the view stayed in the view.
   c. Incorrect. P1 and P2 have different multicast delivery set in V11. Solution: P1 & P2 deliver M32 before V12.
   d. Incorrect. P1 and P2 have different multicast delivery set in V11. Solution: P1 and P2 both deliver their own messages in V11
   e. Correct. Every non-faulty process received the same set of multicasts and everything in the view stayed in the view.
   f. Incorrect. M32 was sent in V11 but delivered in V12. Solution: M32 is delivered to all processes in V11.

10. (Solution and Grading by: <Federico>)

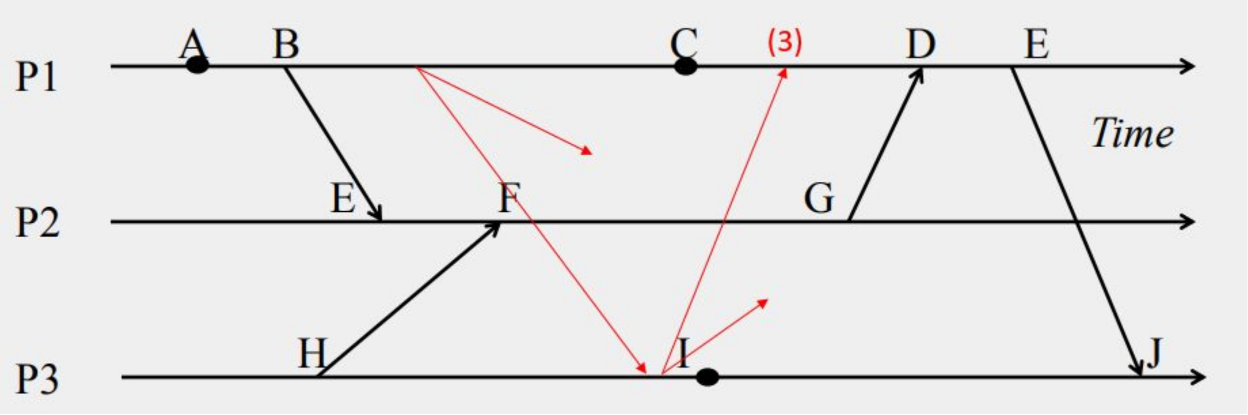
    I. No, this algorithm is not correct. Using the timeline from the lecture slides, here is a counterexample.



At (1), P1 is the initiator so it will record its state S1, send markers to outgoing channels C_{12} and C_{13}, and start recording on incoming channels C_{21}, C_{31}.



At (2), P3 receives its first marker. It'll record its local state S3, mark incoming channel C_{13} as empty, turn on recording for incoming channel C_{23}, then send markers on outgoing channels C_{31} and C_{32}. Next, P3 will record incoming messages on all incoming channels, including C_{13}. Opening the channel that was just marked as closed would allow more messages than necessary to pass through.



At (3), P1 receives a duplicate marker on incoming channel C_{31}, which is only the first (not the (N-1)th or 2nd marker) so it does nothing, **disproving correctness** because the state of C_{31} is never recorded. When the algorithm terminates, the state of C_{31} will not be known, and the snapshot would be incomplete.

Key points in answer

- A channel should be closed as soon as a marker arrives on that channel (not when receiving the last marker on every channel)
- The algorithm waits too long to capture the channel state, and it might capture more messages than necessary in the snapshot

Any counterexample of the algorithm causing an inconsistent cut is also valid.

II. On the last line of the *if* block, change “for j=1 to N except i” to “for j=1 to N except i **and j**” because it should exclude the incoming channel that was already marked (as empty).
In the *else* block, eliminate the if statement because it should not wait until the last marker <u>and</u> eliminate the loop because it will mark all channels C_{ji} (for a given process P_i) as having received messages while some channels may still have messages in transit.

III. (Anything can go here, but no points are earned)