# Royal Institute of Technology

## MSc. Software Engineering of Distributed Systems

ID2203 Distributed Systems Advanced Course

Homework 4

Andrei Shumanski
andreish@kth.se
0046707761992

Trigonakis Vasileios
vtri@kth.se
0046707694420

*Stockholm 2010*

## TABLE OF CONTENTS

## EXERCISE 1

Both Abortable Consensus (Algorithm 2 and Algorithm 3) and Paxos Uniform Consensus (Algorithm 4 and Algorithm 5) use a seen Ids set  to manage different concurrent instances of consensus. The seenIds set grows indefinitely. Explain in your written report, for each algorithm in part, how this set could be garbage collected.

## ANSWER

### PAXOS UNIFORM CONSENSUS

For the PUC algorithm, the garbage collection of seenIds is pretty simple. As soon as a process triggers a ucDecide, it can be sure that all processes will trigger it and, so no process will try to use this seenId again (assuming that processes increase its value before using it for a new instance of the PUC algorithm).

### ABORTABLE CONSENSUS

In this algorithm the solution is not so intuitive. The problem is that this algorithm is used by the upper level algorithm, so not all processes can be sure that an seenId will not be reused. Our solution is:

- only discard the seenId for which a process is sending an acReturn with a non null value to the upper level. That way, the process can be sure that the upper level will never try to use this id again. This solution also lets an increasing seenIds list to most of the nodes.
- if we really want to garbage collect the seenIds to all processes, we need to beBroadcast that we acReturn a non null value to all processes (in the AC level), so they can also get rid of this seenId from their list

## EXERCISE 2

Can PUC be used in a fail-recovery model? If so, under what condition/assumption? If not, why?

## ANSWER

The answer to this question is given into the paper “Consensus: the big misunderstanding form R Guerraoui, A Schiper - Proceedings of the 6th IEEE Computer, 1997 ”. Here we cite the excerpt that answer the question:

*“2. The consensus problem, and the failure detectors, have been defined in a model in which processes do not recover. This simplifies the specification. Take for example a model with process recovery and a process p that crashes and later recovers. Is p a correct process? This is an important question, because the specification of the consensus problem requires that every “correct" process eventually decides. Thus, if the answer is “yes", p must eventually decide; if the answer is “no", p is not obliged to decide. However, the “crash-no recovery" model can be extended to include process recovery. Solving consensus in a “crash recovery" model using failure detectors is discussed in [14]. The solution is very close to the solution in the “crash-no recovery" model.”*

*[14] R. Oliveira, R. Guerraoui, and A. Schiper. Consensus in the Crash-Recover Model. Technical report, EPFL, Dept d'Informatique, July 1997.*

In a nutshell: we use the fail-stop model because it is simpler. If we use a fail-recovery model, we have to use stable storage and also refine our eventual leader election algorithm so that it takes into account the processes' failures and recoveries.

## EXERCISE 3

Construct a topology with 3 processes, use 1000 ms latency between the nodes. Experiment with the following two scenarios:

1:D2200:P1-1
2:D2000:P1-2
3:D2000:P1-3

and

1:D2000:P1-1
2:D2200:P1-2
3:D2200:P1-3

What value is decided for Paxos instance 1 in these two scenarios? Give reasoning in your report why that particular value has been choosen.

## ANSWER

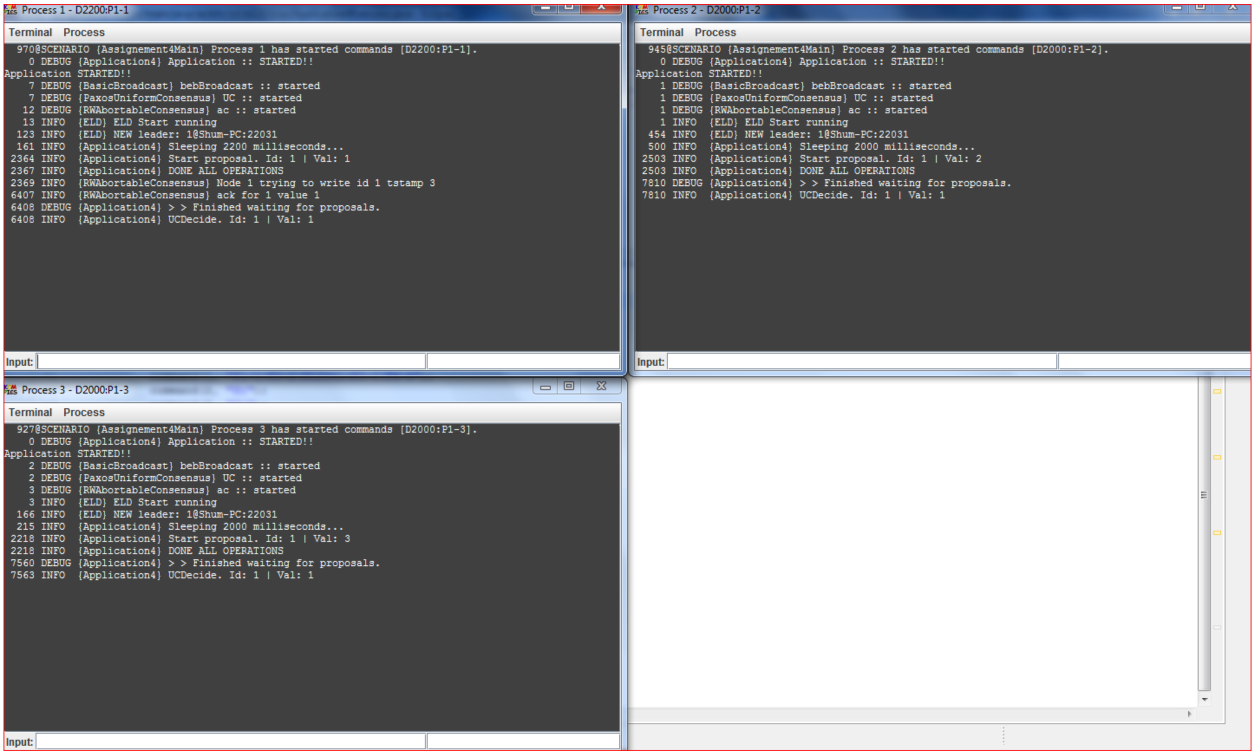
We implemented following scenarios and topology:

```
Topology topologyEx3 = new Topology() {
        {
                node(1, "127.0.0.1", 22031);
                node(2, "127.0.0.1", 22032);
                node(3, "127.0.0.1", 22033);
                defaultLinks(1000, 0);
        }
};

Scenario scenarioEx3_1 = new Scenario(Assignement4Main.class) {
        {;
                command(1, "D2200:P1-1");
                command(2, "D2000:P1-2");
                command(3, "D2000:P1-3");
        }
};

Scenario scenarioEx3_2 = new Scenario(Assignement4Main.class) {
        {;
                command(1, "D2000:P1-1");
                command(2, "D2200:P1-2");
                command(3, "D2200:P1-3");
        }

};
```

We also used following timings for ELD:

```
private static long delta = 1000;

private static long timeDelay = 2000;
```

Results of the first scenario are:

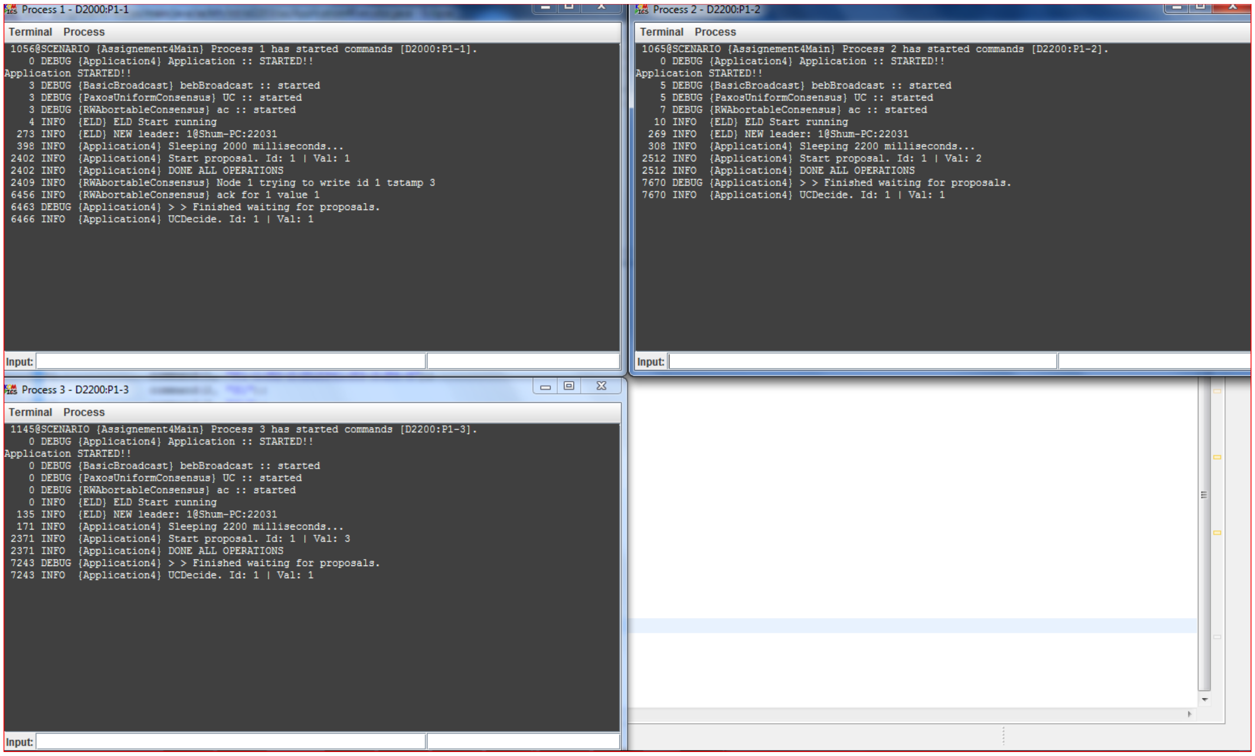


```
  970@SCENARIO {Assignement4Main} Process 1 has started commands [D2200:P1-1].
    0 DEBUG {Application4} Application :: STARTED!!
Application STARTED!!
    7 DEBUG {BasicBroadcast} bebBroadcast :: started
    7 DEBUG {PaxosUniformConsensus} UC :: started
   12 DEBUG {RWAbortableConsensus} ac :: started
   13 INFO  {ELD} ELD Start running
  123 INFO  {ELD} NEW leader: 1@Shum-PC:22031
  161 INFO  {Application4} Sleeping 2200 milliseconds...
 2364 INFO  {Application4} Start proposal. Id: 1 | Val: 1
 2367 INFO  {Application4} DONE ALL OPERATIONS
 2369 INFO  {RWAbortableConsensus} Node 1 trying to write id 1 tstamp 3
 6407 INFO  {RWAbortableConsensus} ack for 1 value 1
 6408 DEBUG {Application4} > > Finished waiting for proposals.
 6408 INFO  {Application4} UCDecide. Id: 1 | Val: 1
```

```
  945@SCENARIO {Assignement4Main} Process 2 has started commands [D2000:P1-2].
    0 DEBUG {Application4} Application :: STARTED!!
Application STARTED!!
    1 DEBUG {BasicBroadcast} bebBroadcast :: started
    1 DEBUG {PaxosUniformConsensus} UC :: started
    1 DEBUG {RWAbortableConsensus} ac :: started
    1 INFO  {ELD} ELD Start running
  454 INFO  {ELD} NEW leader: 1@Shum-PC:22031
  500 INFO  {Application4} Sleeping 2000 milliseconds...
 2503 INFO  {Application4} Start proposal. Id: 1 | Val: 2
 2503 INFO  {Application4} DONE ALL OPERATIONS
 7810 DEBUG {Application4} > > Finished waiting for proposals.
 7810 INFO  {Application4} UCDecide. Id: 1 | Val: 1
```

```
  927@SCENARIO {Assignement4Main} Process 3 has started commands [D2000:P1-3].
    0 DEBUG {Application4} Application :: STARTED!!
Application STARTED!!
    2 DEBUG {BasicBroadcast} bebBroadcast :: started
    2 DEBUG {PaxosUniformConsensus} UC :: started
    3 DEBUG {RWAbortableConsensus} ac :: started
    3 INFO  {ELD} ELD Start running
  166 INFO  {ELD} NEW leader: 1@Shum-PC:22031
  215 INFO  {Application4} Sleeping 2000 milliseconds...
 2218 INFO  {Application4} Start proposal. Id: 1 | Val: 3
 2218 INFO  {Application4} DONE ALL OPERATIONS
 7560 DEBUG {Application4} > > Finished waiting for proposals.
 7563 INFO  {Application4} UCDecide. Id: 1 | Val: 1
```

Results of the second scenario are:



```
 1056@SCENARIO {Assignement4Main} Process 1 has started commands [D2000:P1-1].
    0 DEBUG {Application4} Application :: STARTED!!
Application STARTED!!
    3 DEBUG {BasicBroadcast} bebBroadcast :: started
    3 DEBUG {PaxosUniformConsensus} UC :: started
    3 DEBUG {RWAbortableConsensus} ac :: started
    4 INFO  {ELD} ELD Start running
  273 INFO  {ELD} NEW leader: 1@Shum-PC:22031
  398 INFO  {Application4} Sleeping 2000 milliseconds...
 2402 INFO  {Application4} Start proposal. Id: 1 | Val: 1
 2402 INFO  {Application4} DONE ALL OPERATIONS
 2409 INFO  {RWAbortableConsensus} Node 1 trying to write id 1 tstamp 3
 6456 INFO  {RWAbortableConsensus} ack for 1 value 1
 6463 DEBUG {Application4} > > Finished waiting for proposals.
 6466 INFO  {Application4} UCDecide. Id: 1 | Val: 1
```

```
 1065@SCENARIO {Assignement4Main} Process 2 has started commands [D2200:P1-2].
    0 DEBUG {Application4} Application :: STARTED!!
Application STARTED!!
    5 DEBUG {BasicBroadcast} bebBroadcast :: started
    5 DEBUG {PaxosUniformConsensus} UC :: started
    7 DEBUG {RWAbortableConsensus} ac :: started
   10 INFO  {ELD} ELD Start running
  269 INFO  {ELD} NEW leader: 1@Shum-PC:22031
  308 INFO  {Application4} Sleeping 2200 milliseconds...
 2512 INFO  {Application4} Start proposal. Id: 1 | Val: 2
 2512 INFO  {Application4} DONE ALL OPERATIONS
 7670 DEBUG {Application4} > > Finished waiting for proposals.
 7670 INFO  {Application4} UCDecide. Id: 1 | Val: 1
```

```
 1145@SCENARIO {Assignement4Main} Process 3 has started commands [D2200:P1-3].
    0 DEBUG {Application4} Application :: STARTED!!
Application STARTED!!
    0 DEBUG {BasicBroadcast} bebBroadcast :: started
    0 DEBUG {PaxosUniformConsensus} UC :: started
    0 DEBUG {RWAbortableConsensus} ac :: started
    0 INFO  {ELD} ELD Start running
  135 INFO  {ELD} NEW leader: 1@Shum-PC:22031
  171 INFO  {Application4} Sleeping 2200 milliseconds...
 2371 INFO  {Application4} Start proposal. Id: 1 | Val: 3
 2371 INFO  {Application4} DONE ALL OPERATIONS
 7243 DEBUG {Application4} > > Finished waiting for proposals.
 7243 INFO  {Application4} UCDecide. Id: 1 | Val: 1
```

We can see that in both cases nodes decided on value 1 and it didn’t depend on initial delay. This value was chosen because node 1 has been chosen as a leader and this is the only node that actually tries to write its value. Therefore initial delays do not influence on the execution of the algorithm.

## EXERCISE 4

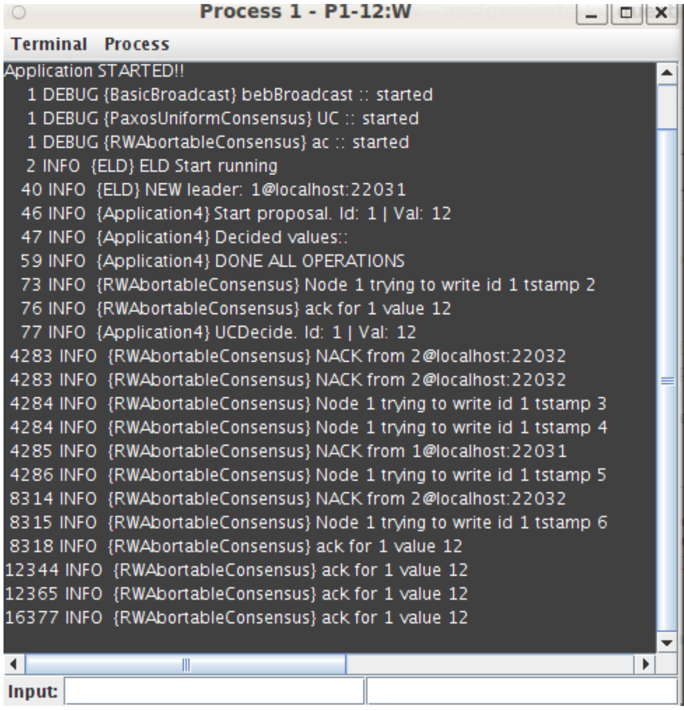
Construct a topology with 2 processes. Assign link delays and initialize ELD period and increment in such a way that the two processes that initiate concurrent PUC proposals abort at least once in AC. Present the topology, the ELD parameter (Timedelay) and the operation sequences (ops) of the two processes and explain the execution.

## ANSWER

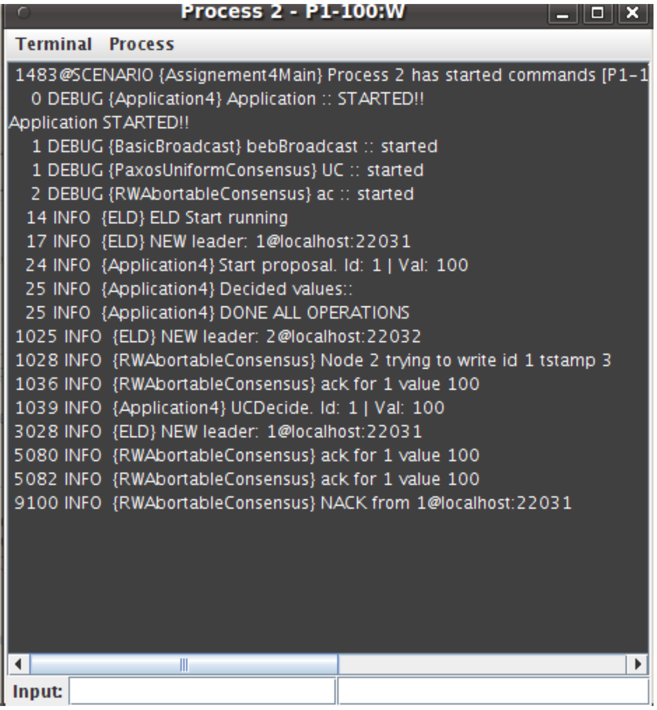
The values used are:

```
Topology topologyQ4 = new Topology() {
        {
                node(1, "127.0.0.1", 22031);
                node(2, "127.0.0.1", 22032);
                defaultLinks(1800, 0);
        }
};


Scenario scenarioQ4 = new Scenario(Assignement4Main.class) {
        {
        command(1, "P1-12:W");
        command(2, "P1-100:W");
        }
};


private static long delta = 1000;
private static long timeDelay = 1000;
```

The resulting execution was:



```
Application STARTED!!
   1 DEBUG {BasicBroadcast} bebBroadcast :: started
   1 DEBUG {PaxosUniformConsensus} UC :: started
   1 DEBUG {RWAbortableConsensus} ac :: started
   2 INFO  {ELD} ELD Start running
  40 INFO  {ELD} NEW leader:  1@localhost:22031
  46 INFO  {Application4} Start proposal. Id:  1 | Val:  12
  47 INFO  {Application4} Decided values::
  59 INFO  {Application4} DONE ALL OPERATIONS
  73 INFO  {RWAbortableConsensus} Node 1 trying to write id 1 tstamp 2
  76 INFO  {RWAbortableConsensus} ack for 1 value 12
  77 INFO  {Application4} UCDecide. Id:  1 | Val:  12
 4283 INFO  {RWAbortableConsensus} NACK from 2@localhost:22032
 4283 INFO  {RWAbortableConsensus} NACK from 2@localhost:22032
 4284 INFO  {RWAbortableConsensus} Node 1 trying to write id 1 tstamp 3
 4284 INFO  {RWAbortableConsensus} Node 1 trying to write id 1 tstamp 4
 4285 INFO  {RWAbortableConsensus} NACK from 1@localhost:22031
 4286 INFO  {RWAbortableConsensus} Node 1 trying to write id 1 tstamp 5
 8314 INFO  {RWAbortableConsensus} NACK from 2@localhost:22032
 8315 INFO  {RWAbortableConsensus} Node 1 trying to write id 1 tstamp 6
 8318 INFO  {RWAbortableConsensus} ack for 1 value 12
12344 INFO  {RWAbortableConsensus} ack for 1 value 12
12365 INFO  {RWAbortableConsensus} ack for 1 value 12
16377 INFO  {RWAbortableConsensus} ack for 1 value 12
```



```
 1483@SCENARIO {Assignement4Main} Process 2 has started commands [P1-1
   0 DEBUG {Application4} Application :: STARTED!!
Application STARTED!!
   1 DEBUG {BasicBroadcast} bebBroadcast :: started
   1 DEBUG {PaxosUniformConsensus} UC :: started
   2 DEBUG {RWAbortableConsensus} ac :: started
  14 INFO  {ELD} ELD Start running
  17 INFO  {ELD} NEW leader:  1@localhost:22031
  24 INFO  {Application4} Start proposal. Id:  1 | Val:  100
  25 INFO  {Application4} Decided values::
  25 INFO  {Application4} DONE ALL OPERATIONS
 1025 INFO  {ELD} NEW leader:  2@localhost:22032
 1028 INFO  {RWAbortableConsensus} Node 2 trying to write id 1 tstamp 3
 1036 INFO  {RWAbortableConsensus} ack for 1 value 100
 1039 INFO  {Application4} UCDecide. Id:  1 | Val:  100
 3028 INFO  {ELD} NEW leader:  1@localhost:22031
 5080 INFO  {RWAbortableConsensus} ack for 1 value 100
 5082 INFO  {RWAbortableConsensus} ack for 1 value 100
 9100 INFO  {RWAbortableConsensus} NACK from 1@localhost:22031
```

## EXPLANATION

1. The nodes start concurrent consensus requests for the same id.
2. Because of the timing delays we chose, the eld of the process 2 makes a false alarm for node 1, and elects process 2 as a leader.
3. The algorithms prerequisites are not correct in this specific moment since we do not have majority of correct processes. Each node is considering itself as leader and ACK its request, resulting that the two nodes, accept their own proposal.

## EXERCISE 5

In the original presentation of the Paxos algorithm, processes can have different roles: proposer, acceptor, and learner. For each event handler of AC and PUC say processes with what role are meant to execute the respective handler. In other words, what types of messages are sent and expected (and what events are triggered and expected) by each process role. Of course one process can act in more than one role. In fact, in our case, processes act in all roles, but interestingly, they need not to. Please refer to lecture 11 and the Paxos paper available on the course webpage.

## ANSWER

For AC:

Upon event ⟨ acPropose | id, v ⟩ do – proposer

Upon event ⟨ bebDeliver | pj ,[Read,id,ts] ⟩ do – acceptor

Upon event ⟨ pp2pDeliver | pj ,[ ,id] ⟩ do – proposer

Upon event ⟨ pp2pDeliver | pj ,[ReadAck,id,ts,v,sentts] ⟩ do – proposer

Upon event ⟨ bebDeliver | pj ,[Write,id,ts,v] ⟩ do – acceptor

uponevent ⟨ pp2pDeliver | pj ,[WriteAck,id,sentts] ⟩ do – proposer

For PUC

Upon event ⟨ trust | pi ⟩ do – proposer

Upon event ⟨ ucPropose | id, v ⟩ do – proposer

Upon event ⟨ acReturn | id,result ⟩ do – proposer

Upon event ⟨ bebDeliver | pi,[Decided,id,v] ⟩ do - learner

# EXERCISE 6

## QUESTION 6.1.

According to the paper, is atomic broadcast (total order broadcast) equivalent to consensus? What does that mean?

### ANSWER

The answer is given into the following excerpt:

*"Such a claim, stating that results applying to the consensus problem are irrelevant to other agreement problems, is incorrect. The simplest example is atomic broadcast: the atomic broadcast problem and the consensus problem have been shown to be equivalent 3]. Equivalence means that (i) any solution to the atomic broadcast problem can be used to solve the consensus problem2, and (ii) any solution to the consensus problem can be used to solve the atomic broadcast problem3 . Because of (i), the atomic broadcast problem is also subject to the FLP impossibility result. Because of (ii), a solution for the consensus problem can be used as a building block to solve the atomic broadcast problem. This last statement is usually turned down with the following argument: it is not because consensus can be used to implement atomic broadcast, that consensus has to be used to implement atomic broadcast. However, because of (i), the inherent difficulty of solving the consensus problem inevitably applies to the atomic broadcast problem, independently of the solution that is used."*

So, in a nutshell: these two problems are equivalent and the result is that they are equivalently difficult to solve in the same environment.

## QUESTION 6.2.

According to the paper, are all notions of time impossible in an asynchronous model?

### ANSWER

The answer is given into the following excerpt:

*"This statement is flawed. The absence of time in the asynchronous system model does not prevent processes from suspecting the crash of other processes.* ***A time-out mechanism can easily be implemented based on a (purely local) logical time:*** *the logical time of process pi can be defined as the number of instructions that pi has executed. In other words, adding time-outs to the asynchronous system model is not enough to overcome the FLP impossibility result 6]. The heart of the FLP result is the impossibility to distinguish crashed processes from those that are slow or connected via slow links."*

So, the outcome is that although talking about time as we are used to is impossible into asynchronous system, we can use the notion of logical times.

## QUESTION 6.3.

Assume we can set the timeout value very high, e.g. 1minute, and that would guarantee that the failure detectors will never suspect a correct node inaccurately. We have then circumvented the FLP impossibility. What practical implications does this have?

### ANSWER

The answer is given into the following excerpt:

*“Nevertheless, there is an inevitable trade-off between (1) reducing the probability of timing failures (i.e., the probability of incorrect failure suspicions), and (2) fast reaction to process crashes. This trade-off is at the heart of the misunderstanding. Consider atomic broadcast implemented using a sequencer process (e.g., Amoeba 12], Isis 1]), and a time-out of 30 seconds to suspect the crash of the sequencer process. In this case, at least 30 seconds are needed to react to the crash of the sequencer process, i.e., the crash of the sequencer will lead to a black-out period of at least 30 seconds. This might be unacceptable for time-critical applications. On the other hand, reducing the time-out value increases the probability of incorrect failure suspicions! The probability of false suspicions might still be low with a time-out of 15 seconds, but it can be high with a time-out of a 1/2 second! As soon as the probability of incorrect failure suspicions becomes non-negligible, it is no longer adequate to consider the system as being synchronous.”*

In short, if we use an 1 min timeout and achieve to guarantee that the failure detectors will never suspect a correct node inaccurately, we actually have a synchronous system! But, this long time out will not be acceptable for more applications.

## QUESTION 6.4.

According to the paper, is it possible to solve consensus in one step?

### ANSWER

The answer is given into the following excerpt:

*“Solving consensus with only one communication step? Solving consensus in two communication steps in good runs is not so bad! Is it possible to do better? Better means solving consensus in a single communication step. If this is not possible, then the early consensus algorithm is optimal in the number of communication steps in good runs! Our intuition is that there is no algorithm, whatever (realistic) system model, even time-based, that solves consensus with a single communication step in good runs . Typically, such an algorithm would decide in good runs on the initial value of one of the processes, e.g., p1 (see Figure 3). However, such an algorithm cannot be correct.”*

So, no, it cannot be solver in one step and, thus the two communication step algorithms are optimal.