



Distributed Systems Part II

Solution to Exercise Sheet 5

1 Paxos Timeline

The timeline consists of two concurrent processes, one on the client Q and one on the client R . In Figure 1 you can see how both clients prepare and propose their values at first, but only the value of client Q gets accepted:

- $T_0 + 0.0$: Q sends a $\text{prepare}(22,1)$. As A and B have never accepted a value they reply with $\text{acc}(\emptyset,0)$.
- $T_0 + 0.5$: R sends a $\text{prepare}(33,2)$. As B and C have never accepted a value they reply with $\text{acc}(\emptyset,0)$.
- $T_0 + 1.0$: Q sends a $\text{propose}(22,1)$. This is acknowledged by A with $\text{ack}(22,1)$ because its $n_{max} = 0$. B does not reply as its value $n_{max} = 2$.
- $T_0 + 2.0$: Q sends a $\text{prepare}(22,3)$. As B has never accepted a value it replies with $\text{acc}(\emptyset,0)$. A returns the latest accepted value: $\text{acc}(22,1)$.
- $T_0 + 2.5$: R sends a $\text{propose}(33,2)$. This is acknowledged by C with $\text{ack}(33,2)$. B does not reply as its value n_{max} is 3.
- $T_0 + 3.0$: Q sends a $\text{propose}(22,3)$. This is acknowledged by A and B with $\text{ack}(22,3)$.
- $T_0 + 4.5$: R sends a $\text{prepare}(33,4)$. C sends back its latest accepted value $\text{ack}(33,2)$. B also sends back its latest accepted value $\text{acc}(22,3)$.
- $T_0 + 6.5$: R sends a $\text{propose}(22,4)$ (It took the newest value from the prepare phase). Both clients B and C reply with an $\text{ack}(22,4)$. All clients have accepted the same value. This means we have achieved consensus.

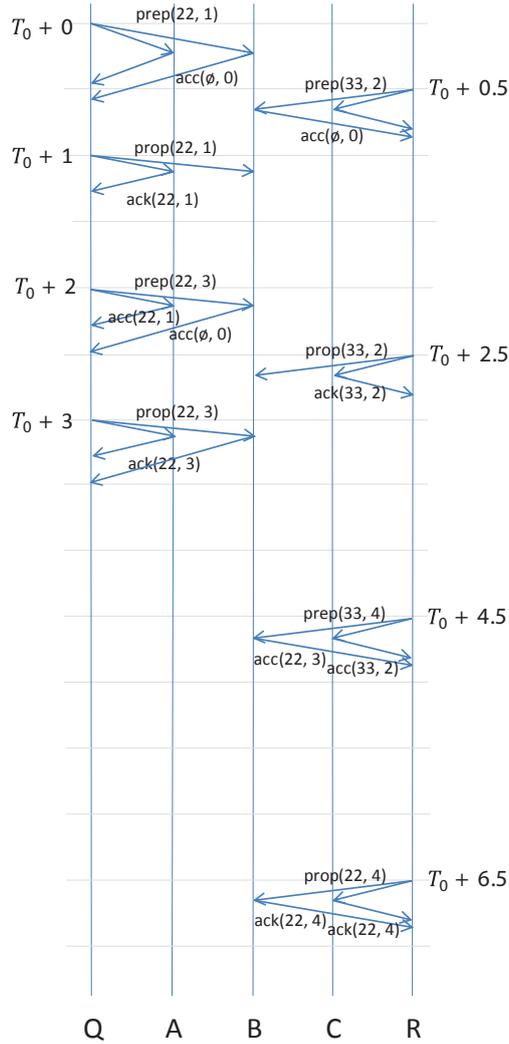


Figure 1: The timeline of the two clients running the given paxos-proposer-program with different timeout values

2 Paxos Acceptors

a) Figure 2 shows an example of how a byzantine client can lead to a failure of the Paxos protocol (i.e. why Paxos is not resilient against byzantine failures):

1. The red proposer sends a prepare with value 1.
2. The red acceptors (incl. the byzantine) send an $\text{ack}(\emptyset, 0)$ back.
3. The blue proposer sends a prepare with its value.
4. The blue acceptors (incl. the byzantine) send an $\text{ack}(\emptyset, 0)$ back. We assume that a read on the faulty register of the byzantine node returned $n_{max} = 0$.
5. The red proposer sends a propose with value 1.
6. The red acceptors (incl. the byzantine) send an $\text{ack}(1, 3)$ back. We assume that a read on the faulty register of the byzantine node returned $n_{max} = 3$.
7. The blue proposer sends a propose with value 1.

8. The blue acceptors (incl. the byzantine) send an $\text{ack}(2,4)$ back. We assume that a read on the faulty register of the byzantine node returned $n_{max} = 4$.

At the end of these 8 steps the red proposer thinks that a majority has accepted the value 1 and the blue proposer thinks that a majority has accepted value 2. Both proposers will start to disseminate their value as each of them thinks that they have achieved consensus.

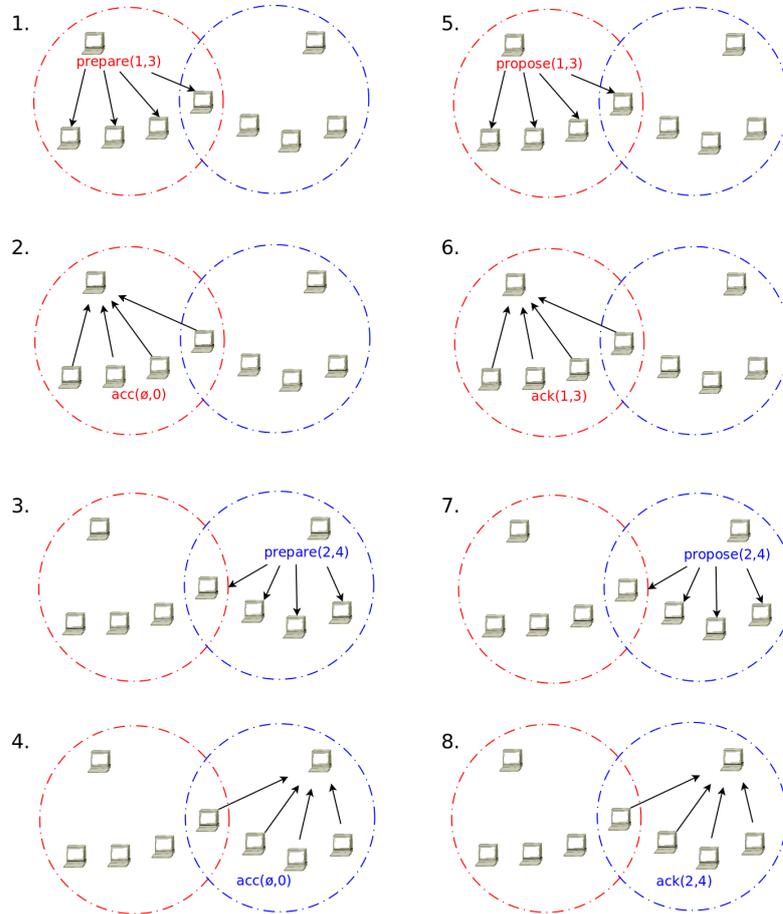


Figure 2: How a byzantine client can lead to different values that are accepted by a majority.

- b) The prepare step allows the proposer and the acceptor to agree on a lower bound of the proposal number that will be accepted. By sending an $\text{ack}(x,y)$ message, the acceptor guarantees the proposer that it will never accept a proposed value that has a smaller timestamp than the one in the prepare message of the proposer.