Introduction to Blockchain Ecosystem: Assignment 1

Sara Baradaran

Question 1

Construct an execution of the PBFT protocol in which the malicious or byzantine leader will affect safety or liveness of the protocol. In other words, construct execution scenarios in which the leader can induce the worst case behavior of the protocol.

Given a request $< REQUEST, o, t, c >$ sent by the client to the primary, first, the primary should send a pre-prepare message to the replicas with a sequence number n for the client request $< REQUEST, o, t, c >$. However, instead of broadcasting a pre-prepare message for the correct request, a malicious primary may either:

- propose an invalid client request $< REQUEST, o', t', c >$, which differs from the request $< REQUEST, o, t, c >$ received from the client.
- propose different requests to different replicas (equivocation), causing a mismatch in what each replica receives.
- ignore the request and does not broadcast any pre-prepare message.

In all the cases above, the client will not receive $f + 1$ valid reply messages corresponding to the request, causing a timeout. Since the pre-prepare messages should include the client's signed request, the PBFT paper assumes that a primary cannot fool non-faulty replicas with an invalid pre-prepare message if it cannot tamper with the client's signature. Thus, a non-faulty node will not broadcast a prepare message for an invalid pre-prepare message and only f faulty nodes might broadcast a prepare for an invalid pre-prepare message. Since each node waits for $2f$ prepare messages to broadcast the commit message, again only faulty nodes may intentionally broadcast commit messages for an invalid request without receiving $2f$ valid prepare messages. In the same way, $2f + 1$ commit messages would not be broadcasted in the network, and thus, a non-faulty node will not execute the operation requested.
All these together, the client does not receive $f + 1$ valid reply messages in a specific time period, resulting in a timeout. In case of a timeout, the client broadcasts the request to all replicas. At this point, non-faulty replicas try to change the view and the primary since the primary has failed to broadcast a valid request. Now, if the primary (or even other faulty nodes) try to flood the network by sending repetitive messages and causing long delays in receiving view change messages, then the protocol may stuck in the view change phase, affecting the liveness of the protocol. Otherwise, if the faulty nodes cannot overwhelm the

network and cause long delays in receiving view change messages by generating excessive network traffic, then view changes will happen after each primary fails to broadcast a valid pre-prepare message. Here, in the worst case, f faulty primaries would be selected consecutively in f rounds of view change, and the protocol finally reaches consensus after $f + 1$ views.

If the malicious primary can tamper with the client's signature, it can also affect the safety of the protocol. Given a valid request $m =< REQUEST, o, t, c >$, the malicious primary can forge a request $m' =< REQUEST, o', t', c >$ $(t << t')$ and broadcast a pre-prepare message for m'. In this scenario, each non-faulty replica, when it receives the pre-prepared message, broadcasts a prepare message. After receiving $2f$ prepare messages with the m''s digest, the valid view and a valid sequence number, each replica will broadcast a commit message. Upon receiving $2f + 1$ valid commits, all the non-faulty nodes will execute o' and send $f + 1$ reply messages to the client with the same result r but with different timestamps from what is requested. Since the client will not receive $f + 1$ reply messages for the m, it broadcasts the request to all replicas, causing a view change to happen. Here, since we assumed $t' >> t$, when the client tries to send his/her requests to a new primary after a view change, the corresponding pre-prepare messages for the next requests would be ignored by non-faulty replicas. This happens since replicas keep the last executed request and do not process pre-prepare messages for a request having timestamp lower than the timestamp of the last request executed. This scenario does not result in inconsistent states among non-faulty replicas. However, this causes an unwanted operation to become executed, and if the client does not process $f + 1$ received reply messages for the fake request, it cannot find why his/her next requests would not be processed by the nodes, even though the next primaries behave well.