1. Mempool staging area needed? If some transactions aren't committed in 5*delta time, should we flush those transactions and let the clients retransmit or should we enqueue them again from the staging area?
There are two types of requests, one when it is sent by the client but not included in the proposal, other one is sent by the client and included in the proposal. Relying on clients for retransmission is not wrong, but no distributed system does it, moreover it can have unexpected problems. So it is better that the system handles it.

2. How do we give a headstart to the system? In main module, a proposal is formed (after forming a qc on the vote msgs) only upon a vote msg (in process_vote_msg). However, assuming we have a committed genesis block and its qc by default, how do we start with creating the proposal? Any better design decisions?
Handle genesis block in a special way since the paper doesn't address the genesis case.

3. Upon a TC, we add the tc in the proposal instead of qc. Now the next proposal will be attached to the highest qc rather than this tc, correct? And if yes, will this tc never have any children? If yes, why do we even bother adding the block with the TC to the block tree?
When a TC is generated, the round is advanced, so essentially, it is not that TC will replace QC. So there's no block which will have a TC and QC = None. If QC was not formed, the proposal will be broadcasted with the highest qc.

4. Can there be forks only at the root or anywhere in the tree? Also, can you give a scenario where forks can happen.
not necessarily from the root. root or one level down. alternating tc and qc for multiple rounds
I guess forks can happen at the root or maybe one level down, but I am not sure. You have to test scenarios to understand that. Maybe alternating TC and QC for multiple rounds is one such scenario which will give more clarity on the branching.

5. Is it safe to use a round-robin fashion with a sufficiently high modulus for pacemaker round increment in diem-bft (This is to avoid increasing the rounds indefinitely). Also, in PBFT with every view change, you update the high and low, but the low is set to the intermediate seq numbr, not 0, so here also the rounds are increasing indefinitely. How is this handled? Is it that when we have a stable checkpoint, we can start from view 0?
You don't have to worry about exhausting round numbers. A 64-bit number will take years to exhaust and by the time the machine would have been changed. However, if you wanna reset the round, make sure you do it carefully

6. What do you exactly mean by syncing up replicas? if a validator gets a block in the proposal msg, it has to extend this block from the high_qc MENTIONED in the block (not its own high_qc, correct?). Now if this validator doesn't have this high_qc it should sync up. This is my understanding of syncing up. Is it correct?
Syncing up means if the validator receives a proposal msg and doesn't have the high qc in the proposal msg it'll get the blocks and sync up.

7. In crypto validity checks, upon arrival of every msg, we check the signature and the sender to verify the sender. However, safety module has its own set of validity checks according to the paper. valid_signatures(b,

last_tc) and valid_signatures(high_qc, last_tc). What are these checks?
Skip the safety checks. It is included when you have safety module on some different machines.

8. Why cannot block_id and ledger state id be the same if they have 1:1 mapping? Or according to the paper, how does ledger state id provide proof of history to client and what's the need; client is concerned only with the replies, once it gets f+1 replies it's happy, where does it need this proof of history? Can we reuse block_id and maybe hash of parent_block_id as state_id.
Another design decision is to create a ledger state tree and in its data field, just maintain pointers to the blocks rather than entire blocks.
It is necessary to have separate blocks and ledger states. Don't worry about the memory overhead because these are very short trees and are pruned once committed. The block id is used for consensus, while the ledger state id is used for verifying the committed states. So it's better to maintain them separately.

9. In leader election, why is there no reputation based leader elected if there's no contiguous chain?
validator leader for a failed round doesn't make sense
If there's no contiguous chain it means that something went wrong and TC was formed because of which continuity was detsroyed. So it would be wrong to elect the validator as a leader when something failed in its reign.

10. On falling back to round-robin, why is it 2 rounds per leader.
The intuition is that if the leader happens to be correct, let it continue for one more round. This is to avoid alternation between correct and faulty leaders.

11. Timeout msgs should be resent if lost. Wouldn't tcp take care of this retransmission? Why do we have to handle this?
You can abstract it with TCP, but what if TCP connection breaks.
However, I (Dhaval) think that this is not the case for us, since we are opening one connection per msg.

12. Let's say we have 4 replicas r1, r2, r3, r4. r1 broadcasts some msg to everyone. However, consider r4 to be crashed at this time. Now r4 won't receive the broadcast. Should r1 keep retrying to send the msg to r4 till it is back again or just let it be?
Don't retry. Consider the case of timeout msg, if a timeout msg is lost and retried again and again, it may timeout some future round unnecessarily when it finally reaches the destination. Better to not do it.