**Meeting Notes 05-04-2022**

**Progress:** Read many papers on distributed concurrency testing: model checking, randomized testing and search-based testing and incorporated feedback into background. I realize that the current method of reordering messages is not enough. Based on the papers and the ripple code, this will likely not expose any concurrency bugs. I would like to get your thoughts on whether I should implement some or all of the following features.

1. Using delays, I feel is not sufficiently concrete enough to cause consistent reordering of messages. I want to try out a more direct scheme where I collect an inbox of enabled events and prioritize the events.
   1. Try this out if I have time. See POS.
2. Inspired by the symmetry used in the FlyMC paper and the identical roles in ripple I would like to assign dynamic node ids based on when the nodes first enter the new consensus round. This should produce better results even with the delay encoding.
   1. Won’t influence much, state symmetry does. Takes too long, don’t implement.
3. I think I can improve this further by changing ripple code. Calls to start round are initiated by a client command or peer message from the scheduler instead of by the node itself.
   1. Try this out.
4. One step further is controlling the timerEntry calls in the individual nodes. Nodes only proceed and send consensus messages at timerEntry calls. Validation messages can occur slightly after due to building the ledger first. Judging by the ripple code, I think postponing messages to after timerEntry calls is also a promising technique and allows.
   1. Leave this for future work.

Ripple bug:

Check updating state on receiving message and try to mess with how it updates that state in the case of asynchronous execution, so lagging and receiving old messages.