CS 7610 - Project 2 Report

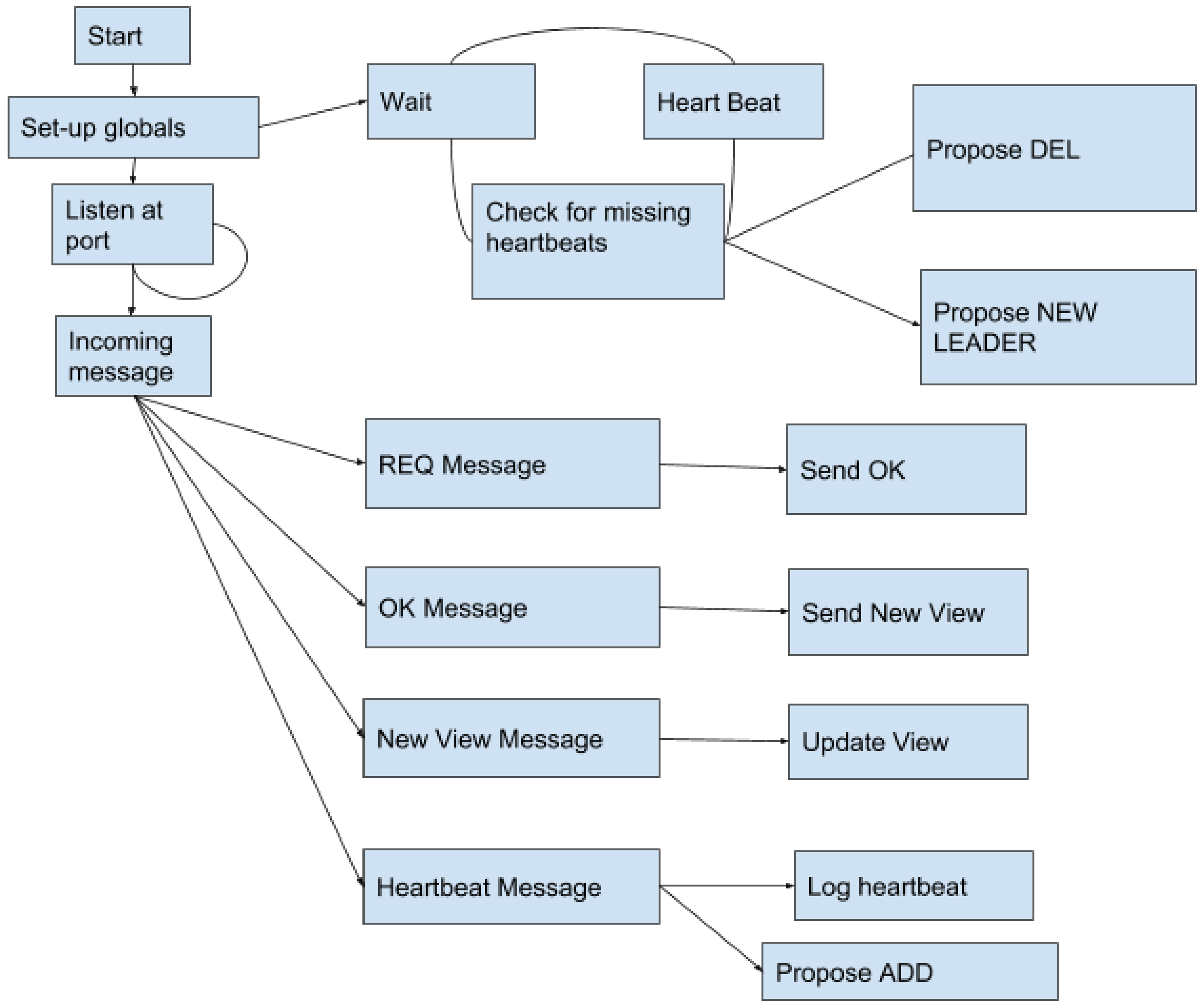
Miles Benjamin

System Architecture

At a high level, each instance of my program sits forever listening at it's port and spawns goroutines (new threads) to handle any incoming traffic. The heartbeat for example is a infinite loop goroutine that sends heartbeat messages to everyone in the view.

The leader is responsible for kicking off a lot of the view change processes, with the exception of the new leader method.

State Diagram



Design Decisions

As you can see the multi-threaded approach makes the core loop of the program very simple. However it does complicate certain aspects of how to check for things that may or may not have happened yet, also how to deal with events that may get double reported before they can be dealt with. However, because my communication is in TCP/IP I don't have to worry about messages getting lost and can assume that messages will either all come in eventually, or something else will change that needs attention first (new things crashing etc.)

Because I chose to rely on heartbeats as the primary form of joining or leaving the group, any missed communication in the ADD or DEL operations would eventually get fixed as long as the heartbeat was still there / missing.

Implementation Issues

This was my first time using golang, which worked out pretty well, although I'm sure there are some things I could have done smarter if I'd had more experience with the language. For example my project has a hard time adding a 4th member to the group, I attribute this to my lack of familiarity with golang as to why I've struggled to debug this and fix it.