Report

Joseph Lewis (lewis285)

Architecture

The whole software is single-threaded, always servicing UDP packets before TCP ones to keep the protocol advancing and so we don't overflow under heavy loads. Luckily with Paxos' quick processing times and no requirement to handle ACK messages things can move quickly.

There are three major portions to the Paxos implementation:

1. The `main.cpp` file which contains the main loop, servicing clients, and passing off messages to the parser.
2. `paxos.cpp` which simply handles the parsing and construction of the Paxos messages. This was mostly automatically generated from `paxos.xml` and `c_generator.py` which converts a packet/protocol definition into structs, a parser, and packet packers. This was also originally built for parsing complex device (e.g. GPS, IMU) protocols in the UDenver Autopilot and re-purposed.
3. `psb.cpp` is the Paxos for System Builders implementation which connects together the callback mechanisms of `paxos.cpp`, and the client/server inputs of `main.cpp` and executes functions via calls to `unicast`.

Design Decisions

External/Other Code

I kept most of the foundational code from the last few projects, modifying my unicast implementation to add in unreliable unicast. I also modified Debug to use vt1000 colors for easier debug output scanning, and the timer originally stolen from my University of Denver autopilot to support the kind of timeouts we need.

Again I kept using the excellent `optionparser` library to do option parsing because that is one of the fundamentally hard problems in CS that never seems to be done right the first time around.

I also utilized the dyad library for constructing TCP client/servers because of its nifty callback interface, ability to support multiple clients, utilization of the `select` system call, and ability to integrate well into the main loop of the program (didn't need threading and synchronization to work).

Implementation Issues

There were quite a few places where the paper seemed contradictory, inconsistent, and incomplete despite in its intro claiming to be a complete implementation. Luckily there are quite a few implementations online albeit in Java/Scala that I could consult to figure out the ambiguities. Interestingly one of these used the bully algorithm for leader election which I rather liked as it seems quite a bit simpler than what we were doing.

State Diagrams