Programming Project 2 for Distributed Systems (ECE 499/599 / CS 419/519)

1 Overview

The learning objectives of this project are i) to learn about the challenges of implementing a distributed leader election algorithm, and ii) to learn the Go programming language. After completing the assignment, you should have a good understanding and appreciation of challenges involved in distributed leader election protocols, and should be able to build distributed systems using Go language's concurrency primitives.

2. Submission Guidelines

You need to submit code implementing the leader election protocol of "Raft" (https://raft.github.io) and then demo your code.

3 Environment and Tasks

Installing Go Programming Language: In this MP, you will use the Go Programming Language (https://golang.org/doc/). Please install the latest version of Go compatible with your platform and the necessary libraries and files (https://golang.org).
You are allowed to implement it in Python too.

3.1 [100 pts] **Raft Leader Election:** Implement the leader election sub-protocol of Raft consensus algorithm. The implementation should be able to detect the failure of the current leader and elect a new leader. Please make good use of the excellent documentation available for Raft algorithm here -- https://raft.github.io.

An extended version of the original Raft paper (https://raft.github.io/raft.pdf) is valuable, as is the nice interactive visualization of the algorithm (http://thesecretlivesofdata.com/raft/). There are many implementations of Raft including in Go language. Some of them are linked through the Raft page. You are welcome to browse the code and learn from as many implementations as you wish. But the code submitted for this project has to be your own.

In order to see and evaluate the functionality of your code, each server has to log its significant actions (e.g. leader failure detected, election timer started, initiated election, received request for votes, etc). The log can be printed to screen/terminal or buffered to a file. Servers have to communicate using Go's RPC not through shared variable or files.

Grading:
Program compiles and runs without failure: 30 points
Passes two test cases: 70 points

Test case 1: When the servers are started, they initiate a leader election algorithm, and agree on a leader. This should be evident from observing the logs (make sure you generate a log for all significant events, for example, including heartbeats) that each server generates.

Test case 2: If the existing leader crashes, the rest of the servers quickly detect the failure, initiate a leader election algorithm, and agree on a new leader. This should be evident from observing the logs (make sure you generate a log for all significant events, for example, including heartbeats) that each server generates.

Please provide a bash file to run your Code. Also provide a README file to explain inputs (# of nodes, crash simulation) and outputs (Logs and events printed in commandline)