Lab 2A: Raft Leader Election

Disclaimer

- I couldn't guarantee the correctness of my solution: passed the tests for >10 times with no race conditions.

Hints on the spec

- There are a few hints on the spec that requires deliberate design

> This way a peer will learn who is the leader, if there is already a leader, or become the leader itself.

- This essentially is alluding to the `ticker()` function. However, I reorganized my `ticker()` to also let it include the logic of the Leader
- Notice that I unlock both before `rf.AttemptElection` and `rf.AttemptAppend`, the lock should not be held when making RPC calls

```
// The ticker go routine starts a new election if this peer hasn't received
// heartsbeats recently.
func (rf *Raft) ticker() {
  for !rf.killed() {
    // Your code here to check if a leader election should
    // be started and to randomize sleeping time using
    // time.Sleep().
    rf.mu.Lock()
    if rf.state != Leader {
      electionTimeout := time.Duration(200+rand.Intn(150)) * time.Millisecond
      if time.Since(rf.lastReceive) > electionTimeout {
        rf.mu.Unlock()
        rf.AttemptElection()
      } else {
        rf.mu.Unlock()
      }
    } else {
      rf.mu.Unlock()
      rf.AttemptAppend()
    }
```

```
        time.Sleep(10 * time.Millisecond)
    }
}
```

> AppendEntries RPC struct should be sent out to reset election timeout so that other servers don’t step forward as leaders when one has already been elected

- A simple method to keep track of election timeout is to initialize a timer for every raft. I called mine: `rf.lastReceive`
- Timer Update rule: The *Student’s Guide to Raft* mentions that
    - “ Specifically, you should *only* restart your election timer if a) you get an `AppendEntries` RPC from the *current* leader (i.e., if the term in the `AppendEntries` arguments is outdated, you should *not* reset your timer); b) you are starting an election; or c) you *grant* a vote to another peer.”
    - This tip is super useful and is basically how I implemented my `rf.lastReceive`. You must implement the timer like this

Students' Guide to Raft

For the past few months, I have been a Teaching Assistant for MIT's 6.824 Distributed Systems class. The class has traditionally had a number of labs building on the Paxos consensus algorithm, but this year, we decided to make the move to Raft.

https://thesquareplanet.com/blog/students-guide-to-raft/

Mistakes I made

- Race Conditions
    - I started off not using `-race` when running my code. It turns out that I forgot to lock several places when using `rf`. A rule of thumb I found at least for this lab is to lock whenever you access a `rf` state
- Variable Initializations
    - The Raft at initial step should be set as the following

```
rf := &Raft{
    me:          me,
    peers:       peers,
    dead:        0,
    state:       Follower, // Forgot this at very beginning
    votedFor:    -1, // Forgot this at very begining
    lastReceive: time.Now(), // Forgot this after a while
}
```

Debugging Tips

- The implementation is rather easy, but it took me way more time to debug than that of implementation
- Make sure you refer to the paper and every single statement on the paper should be treated as MUST
- These tips are mentioned in Lab 5 and I found them super helpful
    - DPrintf: in `util.go`, there is a helper function printing meaningful log. I always redirect the output to a `log.txt` and analyze my log when weird things happen
    - race detector: `-race` saved my life