Separate Code Per State

```cpp
void Server::run() {
    deque<RaftMessage> mail_pile;
    server_loop: while(true) {
        switch (state) {
        case LEADER: {
            cout << "Leader!" << endl;
            RaftMessage request_append_entries = new_raft_message();
            request_append_entries.set_type_of_message(
                RaftMessage::REQUEST_APPEND_ENTRIES);
            broadcast_raft_message(request_append_entries);

            auto time_of_next_heartbeat = get_leader_heartbeat_timeout_time();
            while (milliseconds_until_time(time_of_next_heartbeat) > 0) {
                // deal with mail pile
                while (!mail_pile.empty()) {
                    RaftMessage first_msg = mail_pile.front();
                    if (first_msg.curr_term() > storage.get_curr_term()) {
                        update_server_term(first_msg.curr_term());
                        state = FOLLOWER;
                        goto server_loop;
                    } else if (first_msg.curr_term() < storage.get_curr_term()) {
                        switch (first_msg.type_of_message()) {
                        case RaftMessage::REQUEST_VOTE: {
                            RaftMessage response = new_raft_message();
                            response.set_type_of_message(
                                    RaftMessage::RESPONSE_VOTE);
                            response.set_vote_granted(false);
                            send_raft_message(response, first_msg.server_id());
                        break;
                        }
                        case RaftMessage::REQUEST_APPEND_ENTRIES: {
                            RaftMessage response = new_raft_message();
                            response.set_type_of_message(
                                RaftMessage::RESPONSE_APPEND_ENTRIES);
                            response.set_entries_appended(false);
                            send_raft_message(response, first_msg.server_id());
                        break;
                        }
                        default: {
                        break;
                        }
                        }
                        mail_pile.pop_front();
                    } else {
                        mail_pile.pop_front();
                    }
                }
                // add to mail pile
                int heartbeat_timeout = milliseconds_until_time(
                        time_of_next_heartbeat);
                if (heartbeat_timeout > 0) {
                    vector<int> readable_ids;
                    if (mailbox.wait_for_mail((size_t) heartbeat_timeout,
                            readable_ids)) {
                        add_mail_to_pile(mailbox, readable_ids, mail_pile);
                    }
                }
            }
        break;
        }
```

```cpp
case CANDIDATE: {
    cout << "Candidate!" << endl;
    update_server_term(storage.get_curr_term() + 1);
    cout << "Current Term: " << storage.get_curr_term() << endl;

    // track votes and vote for self
    unordered_set<int> votes_granted_from_ids;
    storage.set_voted_for(self_id);
    votes_granted_from_ids.insert(self_id);

    auto time_of_election_end = get_election_timeout_time();

    // send request_vote to all other servers
    RaftMessage request_vote = new_raft_message();
    request_vote.set_type_of_message(RaftMessage::REQUEST_VOTE);
    broadcast_raft_message(request_vote);

    while (milliseconds_until_time(time_of_election_end) > 0) {
        // deal with mail pile
        while (!mail_pile.empty()) {
            RaftMessage first_msg = mail_pile.front();
            if (first_msg.curr_term() > storage.get_curr_term()) {
                update_server_term(first_msg.curr_term());
                state = FOLLOWER;
                goto server_loop;
            } else if (first_msg.curr_term() < storage.get_curr_term()) {
                switch (first_msg.type_of_message()) {
                case RaftMessage::REQUEST_VOTE: {
                    RaftMessage response = new_raft_message();
                    response.set_type_of_message(RaftMessage::RESPONSE_VOTE);
                    response.set_vote_granted(false);
                    send_raft_message(response, first_msg.server_id());
                    break;
                }
                case RaftMessage::REQUEST_APPEND_ENTRIES: {
                    RaftMessage response = new_raft_message();
                    response.set_type_of_message(
                        RaftMessage::RESPONSE_APPEND_ENTRIES);
                    response.set_entries_appended(false);
                    send_raft_message(response, first_msg.server_id());
                break;
                }
                default: {
                    break;
                }
                }
                mail_pile.pop_front();
            } else {
                if (first_msg.type_of_message()
                        == RaftMessage::RESPONSE_VOTE) {
                    cout << "Reading mail from " << first_msg.server_id() << " vote granted: "
                            << first_msg.vote_granted() << endl;
                }
                if ((first_msg.type_of_message()
                        == RaftMessage::RESPONSE_VOTE)
                    && first_msg.vote_granted()) {
                    votes_granted_from_ids.insert(first_msg.server_id());
                    if ((2 * votes_granted_from_ids.size()) > num_servers) {
                        mail_pile.pop_front();
                        state = LEADER;
                        goto server_loop;
                    }
                // append entries RPC received from new leader
                } else if (first_msg.type_of_message()
                    == RaftMessage::REQUEST_APPEND_ENTRIES) {
                    state = FOLLOWER;
                    goto server_loop;
                }
                mail_pile.pop_front();
            }
        }
        // add to mail pile
        int election_timeout = milliseconds_until_time(time_of_election_end);
        if (election_timeout > 0) {
            vector<int> readable_ids;
            if (mailbox.wait_for_mail((size_t) election_timeout, readable_ids)) {
                add_mail_to_pile(mailbox, readable_ids, mail_pile);
            }
        }
    }

    // check if we should become leader despite not receiving votes
    // occurs if there is only one server in the cluster
    if ((2 * votes_granted_from_ids.size()) > num_servers) {
        state = LEADER;
    }
break;
}
```

```cpp
case FOLLOWER: {
    cout << "Follower!" << endl;
    auto next_heartbeat_deadline = get_election_timeout_time();
    cout << "Election timeout milliseconds: " <<
            milliseconds_until_time(next_heartbeat_deadline) << endl;

    while (milliseconds_until_time(next_heartbeat_deadline) > 0) {
        while (!mail_pile.empty()) {
            RaftMessage first_msg = mail_pile.front();
            if (first_msg.curr_term() > storage.get_curr_term()) {
                update_server_term(first_msg.curr_term());
                state = FOLLOWER;
                goto server_loop;
            } else if (first_msg.curr_term() < storage.get_curr_term()) {
                switch (first_msg.type_of_message()) {
                case RaftMessage::REQUEST_VOTE: {
                    RaftMessage response = new_raft_message();
                    response.set_type_of_message(RaftMessage::RESPONSE_VOTE);
                    response.set_vote_granted(false);
                    send_raft_message(response, first_msg.server_id());
                    break;
                }
                case RaftMessage::REQUEST_APPEND_ENTRIES: {
                    RaftMessage response = new_raft_message();
                    response.set_type_of_message(
                        RaftMessage::RESPONSE_APPEND_ENTRIES);
                    response.set_entries_appended(false);
                    send_raft_message(response, first_msg.server_id());
                    break;
                }
                default: {
                    break;
                }
                }
                mail_pile.pop_front();
            } else { //first_msg.curr_term() == current term
                switch (first_msg.type_of_message()) {
                case RaftMessage::REQUEST_VOTE: {
                    RaftMessage response = new_raft_message();
                    response.set_type_of_message(RaftMessage::RESPONSE_VOTE);
                    // determine how to vote
                    if (storage.get_voted_for() == -1
                        || storage.get_voted_for() == first_msg.server_id()) {
                        storage.set_voted_for(first_msg.server_id());
                        response.set_vote_granted(true);
                    } else {
                        response.set_vote_granted(false);
                    }

                    send_raft_message(response, first_msg.server_id());
                    cout << "Voted for " << storage.get_voted_for() << " in current term "
                            << storage.get_curr_term() << endl;
                    cout << "Sent mail to " << first_msg.server_id() << " vote granted: "
                            << response.vote_granted() << endl;
                    if (response.vote_granted()) {
                        mail_pile.pop_front();
                        state = FOLLOWER;
                        goto server_loop;
                    }
                break;
                }
                case RaftMessage::REQUEST_APPEND_ENTRIES: {
                    RaftMessage response = new_raft_message();
                    response.set_type_of_message(
                        RaftMessage::RESPONSE_APPEND_ENTRIES);
                    response.set_entries_appended(false);

                    send_raft_message(response, first_msg.server_id());
                    mail_pile.pop_front();
                    state = FOLLOWER;
                    goto server_loop;
                break;
                }
                // all other cases
                case RaftMessage::RESPONSE_VOTE:
                case RaftMessage::RESPONSE_APPEND_ENTRIES:
                default:
                    break;
                }
                mail_pile.pop_front();
            }
        }
        // add to mail pile
        int heartbeat_timeout = milliseconds_until_time(next_heartbeat_deadline);
        if (heartbeat_timeout > 0) {
            vector<int> readable_ids;
            if (mailbox.wait_for_mail(heartbeat_timeout, readable_ids)) {
                add_mail_to_pile(mailbox, readable_ids, mail_pile);
            }
        }
    }
    state = CANDIDATE;
break;
}
```

Dispatch on Message Type

```
void RaftServer::HandlePeerMessage(Peer* peer, char* raw_message, int raw_message_len) {
    lock_guard<mutex> lock(server_mutex);
    PeerMessage message;
    message.ParseFromString(string(raw_message, raw_message_len));

    LOG(DEBUG) << "RECEIVE: " << Util::ProtoDebugString(message);

    if (message.term() > storage.current_term()) {
        TransitionCurrentTerm(message.term());
        TransitionServerState(Follower);
    }

    switch (message.type()) {
        case PeerMessage::APPENDENTRIES_REQUEST:
            if (message.term() < storage.current_term()) {
                SendAppendEntriesResponse(peer, false);
                return;
            }
            SendAppendEntriesResponse(peer, true);
            election_timer->Reset();
            return;

        case PeerMessage::APPENDENTRIES_RESPONSE:
            if (message.term() < storage.current_term()) {
                // Drop responses with an outdated term; they indicate this
                // response is for a request from a previous term.
                return;
            }
            return;

        case PeerMessage::REQUESTVOTE_REQUEST:
            if (message.term() < storage.current_term()) {
                SendRequestVoteResponse(peer, false);
                return;
            }
            if (storage.voted_for() != "" &&
                storage.voted_for() != message.server_id()) {
                SendRequestVoteResponse(peer, false);
                return;
            }
            storage.set_voted_for(message.server_id());
            SendRequestVoteResponse(peer, true);
            election_timer->Reset();
            return;

        case PeerMessage::REQUESTVOTE_RESPONSE:
            if (message.term() < storage.current_term()) {
                // Drop responses with an outdated term; they indicate this
                // response is for a request from a previous term.
                return;
            }
            if (message.vote_granted()) {
                ReceiveVote(message.server_id());
            }
            return;

        default:
            cerr << oslock << "Unexpected message type: " <<
                Util::ProtoDebugString(message) << endl << osunlock;
            throw RaftServerException();
    }
}
```

```
void RaftServer::TransitionServerState(ServerState new_state) {
    LOG(INFO) << "STATE: " << getServerStateString(server_state) <<
        " -> " << getServerStateString(new_state);

    server_state = new_state;

    switch (new_state) {
        case Follower:
            return;

        case Candidate:
            TransitionCurrentTerm(storage.current_term() + 1);

            // Candidate server votes for itself
            storage.set_voted_for(server_id);
            ReceiveVote(server_id);

            for (Peer* peer: peers) {
                SendRequestVoteRequest(peer);
            }
            election_timer->Reset();
            return;

        case Leader:
            return;

        default:
            cerr << oslock << "Bad state transition to " << new_state << endl <<
                osunlock;
            throw RaftServerException();
    }
}

void RaftServer::ReceiveVote(string server_id) {
    votes[server_id] = true;
    if (server_state == Leader) return;

    int vote_count = 0;
    for (auto const& [_, vote_granted]: votes) {
        if (vote_granted) vote_count += 1;
    }

    int server_count = peers.size() + 1;
    int majority_threshold = (server_count / 2) + 1;
    if (vote_count >= majority_threshold) {
        TransitionServerState(Leader);
    }
}
```