```
// server variables
currentTerm
votedFor
log[]
commitIndex
lastApplied
nextIndex[]
matchIndex[]
myState                         // this server state {candidate, follower, leader}

// Operations

AppendEntriesReq ( term, leaderId, prevLogIndex, prevLogTerm, entries[], leaderCommit ) ( term,
success ) {
        if ( term < currentTerm ) {
                // This node is not so modern
                // or election has started and this server is in candidate state
                // In all states applicable
                return currentTerm, false
        }

        else if ( term >= currentTerm ) {
                // mystate == leader && term == currentTerm, this is impossible, as no two leaders
will be elected at any term
                if ( mystate == leader && term == currentTerm ) {
                        return null, false
                }

                // Reset heartbeat timeout
                Alarm ( heartbeat_time )

                // This server term is not so up-to-date, so update
                // Convert to follower if current state is candidate/leader
                myState = follower
                currentTerm = term

                // Rest is for follower only
                else if ( log[prevLogIndex] == null || log[prevLogIndex].Term != prevLogTerm ) {
                        // Prev msg index,term doesn't match, i.e. missing previous entries, force
leader to send previous entries
                        return currentTerm, false
                }

                if( log[prevLogIndex+1] != null && log[prevLogIndex+1].Term != term ) {
                        // There are garbage entries from last leaders
                        // Strip them up to the end
                        log.Trim( from(prevLogIndex+1), to(end) )
                }
                // Update log if entries are not present
                log.Append ( term, entries )

                if ( leaderCommit > commitIndex ) {
                        // If leader has commited entries, so should this server
                        commitIndex = min(leaderCommit, log.length)
                }
        }

        return currentTerm, true
}

AppendEntriesResp ( term, success ) {
        if ( term == null ) {
                // Invalid request to leader with same term
                // Impossible state of system, two leaders elected at same term
                myState = follower OR crash or something?
                return
        }

        if ( currentTerm < term ) {
                // There is another latest leader, so back to the follower state
                currentTerm = term
                myState = follower
                return
        }

        if ( success == false ) {
                // Decrease nextIndex
                nextIndex[serverId]--
                // retry with older entries
                return
        }
        else {
                // If success
                <increment number of responses for a given entries>

                for ( i=lastApplied+1 ; i<log.length ; i++ ) {
                        // Starting from lastApplied index
                        if ( log[i].replicas > #_of_nodes/2 ) {
                                // If any entry is on majority of nodes, commit all entries up to prev
entry of that
                                lastApplied = i-1
                        }
                        else {
                                // if there is hole, ignore all rest commiting
                                break
```

```
                    }
                }
                nextIndex[serverId] += entries.length
                matchIndex[serverId] += entries.length
        }

}

RequestVote ( term, candidateId, lastLogIndex, lastLogTerm ) ( term, voteGranted ) {
        if ( term < currentTerm ) {
                // In any state, if old termed candidate request vote, tell it to be a follower
                return currentTerm, false
        }

        // If this state is candidate, this server must have had voted for self
        if ( votedFor == null ) {
                // if leader or follower
                if ( lastLogTerm > log.LastEntry.Term || lastLogTerm == log.LastEntry.Term &&
lastLogIndex > log.length ) {
                        votedFor = candidateId
                        currentTerm = term
                        return currentTerm, true
                }
        }

        // else if candidate and term==currentTerm, reject vote
        return currentTerm, false
}

RequestVoteResp ( term, voteGranted ) {
        if ( voteGranted ) {
                increment # of votes
                if ( #_of_votes > size_of_raft/2 ) {
                        myState = leader
                        Reset ( nextIndex, value=log.length )
                        Reset ( matchIndex, value=0 )
                        Send ( to_all, AppendEntries(empty) )
                }
        }
        else if ( term > currentTerm ) {
                // Overall RAFT is more modern than this system
                currentTerm = term
                myState = follower
        }
}
```

```
Timeout () {
        if ( mystate == candidate ) {
                // if candidate: election slot timeout
                // Restart election
                myState = candidate
        }

        else if ( myState == leader ) {
                // Send heartbeat reminder
                Send( to_all, AppendEntries(empty) )
                return
        }

        else if ( myState == follower ) {
                // Heartbeat timeout
                myState = candidate
        }

        votedFor = selfId
        Alarm ( election_time )
        Send ( to_all , RequestVote )
}
```