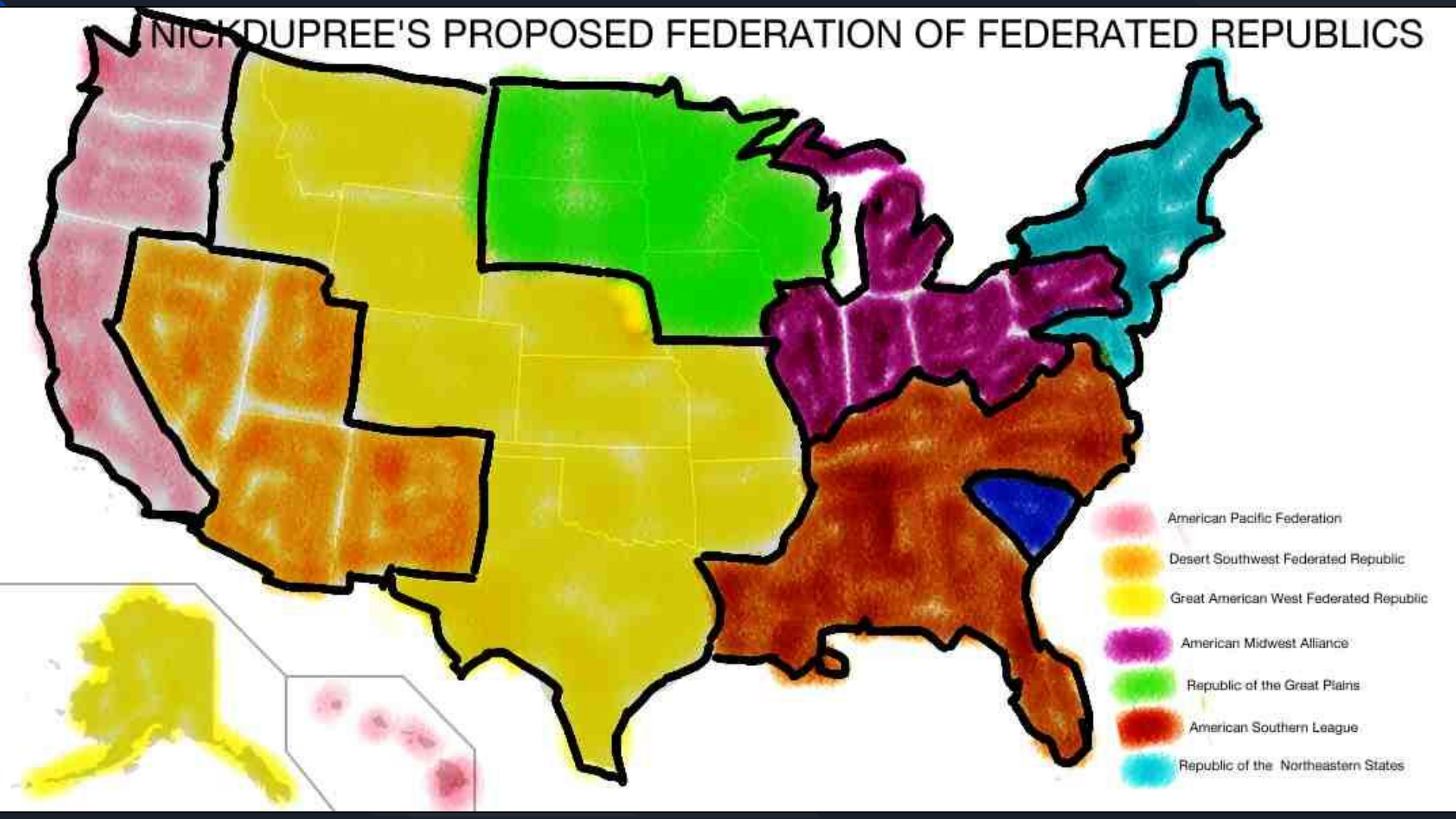
Federated Election Count -Shivam Patel

- A large interest of mine from when i was a child was p2p computing/networks (think limewire, piratebay, mediafire)
- As the years have gone by, the media companies have realized theyll destroy that market by offering cheap streaming services. And for the most part they have won
- Blockchain/Smart contract land will scratch the old itch for anyone interested(ICP, ETH, XMR, AVAX, IPFS, FILECOIN...)
- Neither the p2p/Federated/distributed networks of the past, nor the ones now, have managed to come up with an optimal theory of governance comparable to what real life governments have.
- Lets try to take baby step 0.

NICKDUPREE'S PROPOSED FEDERATION OF FEDERATED REPUBLICS



Implementation:

- Topology is one of 5 "Regional" hosts and a "Citizen" host connected to a central switch called the Coordinator, that helps prepare messages on behalf of the 5 Regions. It does so by marking the packets with phase numbers, and as well keeps track of how many branches of the legislature it has received responses from. If 2/3 legislatures respond, its forwarded to a broadcaster host to broadcast to the 5 regions and the citizen.
- The 3 switches that handle verifying votes (logging them so that in a future phase 2/3 can be reached, as well drop duplicate) are called the legislature. If this implementation was proper, then 1/3 of them could fail and the results of the elections would still be the same.
- Use registers in p4, to handle the collective "state" of the collective 50 states to the best ability possible. If the multicast mechanism was more straightforward, this implementation would have been a full success rather than a partial, due to the fact the problems are coming from interactions with the broadcaster host, and hosts at the back end of the legislature switches.

Results

- The results were ok but not perfect.
- About 80% success in the sense that due to high network congestion some packets were lost.
- As well, Since hosts has to artificially be used where they ideally shouldn't have been(for example the broadcaster maintaining state, due to conditional logic not being allowed in our p4 environment inside of actions, as well that the broadcaster host had to be used at all rather than multi cast group features in p4 due to time constraints) It appears to be the case that since the hosts had to send/receive similtanously, and threads were used. There were numerous problems regarding their virtualization, pythons GIL, and other things.
- Result was some results being hidden due to the threads competing for the GIL, as well as network congestion and packet loss due to the volume of uneccesary steps.

Possible improvements

- Under the assumption that the issues are coming from having to run so many multi threaded hosts concurrently(extra concurrent) as well as the high volume of network communication each one has to due, as well as pythons limitations regarding the GIL, it is reasonable to believe the concurrency, latency, and data loss problems could be solved by moving the features of the broadcaster into the coordinator switch or attaching some new types of switches.
- For the parts that have to be implemented in python, use multiprocessing rather than multi threading. There is a bit more complication but not alot since its mainly IO tasks we're doing, and state can be shared when needed with signals and pipes. The benefit of this is, we are guaranteed that so long as number of cores are there, we dont have processes fighting eachother over who can print/write to files and IO.