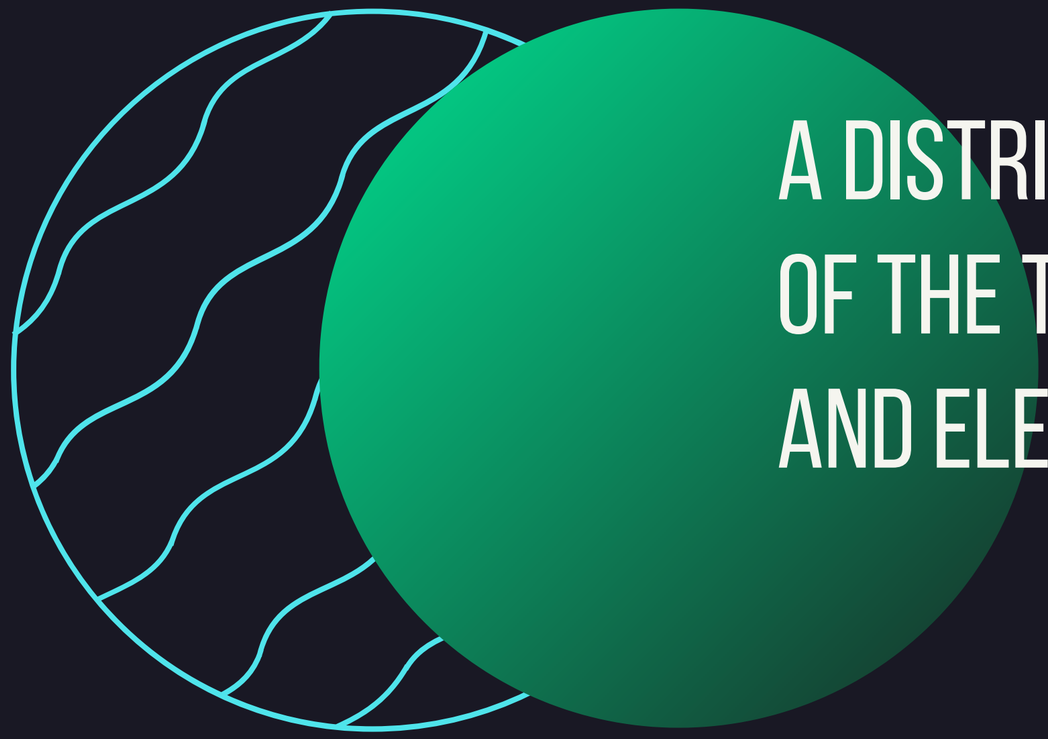
COMP 755 - FALL 2021



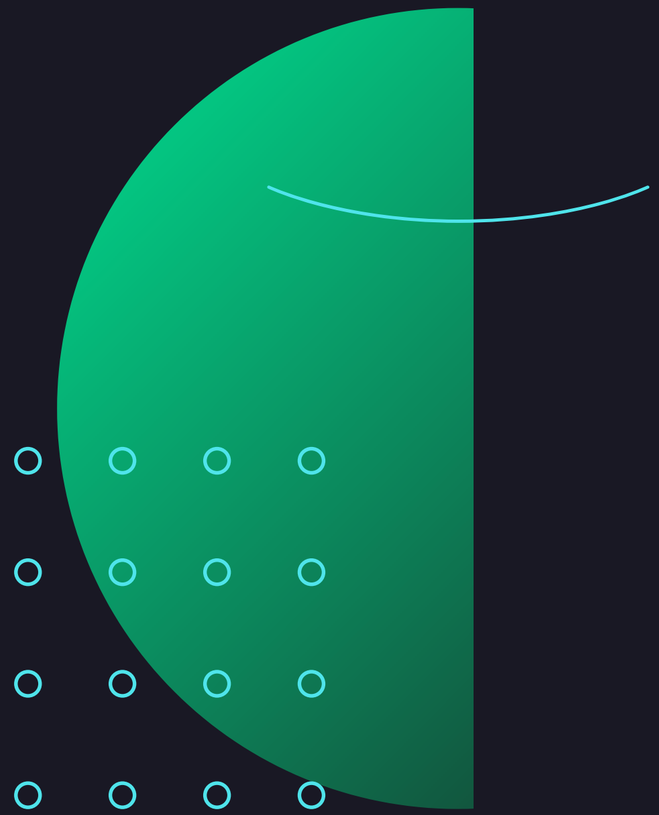
A DISTRIBUTED IMPLEMENTATION OF THE TEMPO BERKELEY ALGORITHM AND ELECTION ALGORITHM

BY: JORDAN A. STURTZ

Outline

- Goals of the Project
- Background
- Related Work
- Implementation
- Challenges
- Q&A

PROJECT GOALS



Implement A Clock Synchronization Algorithm

- Gusella and Zatti's "Berkeley" Algorithm

Implement an Election Algorithm

- Gusella and Zatti's Election Algorithm

Measure Performance

- Check correctness

BACKGROUND

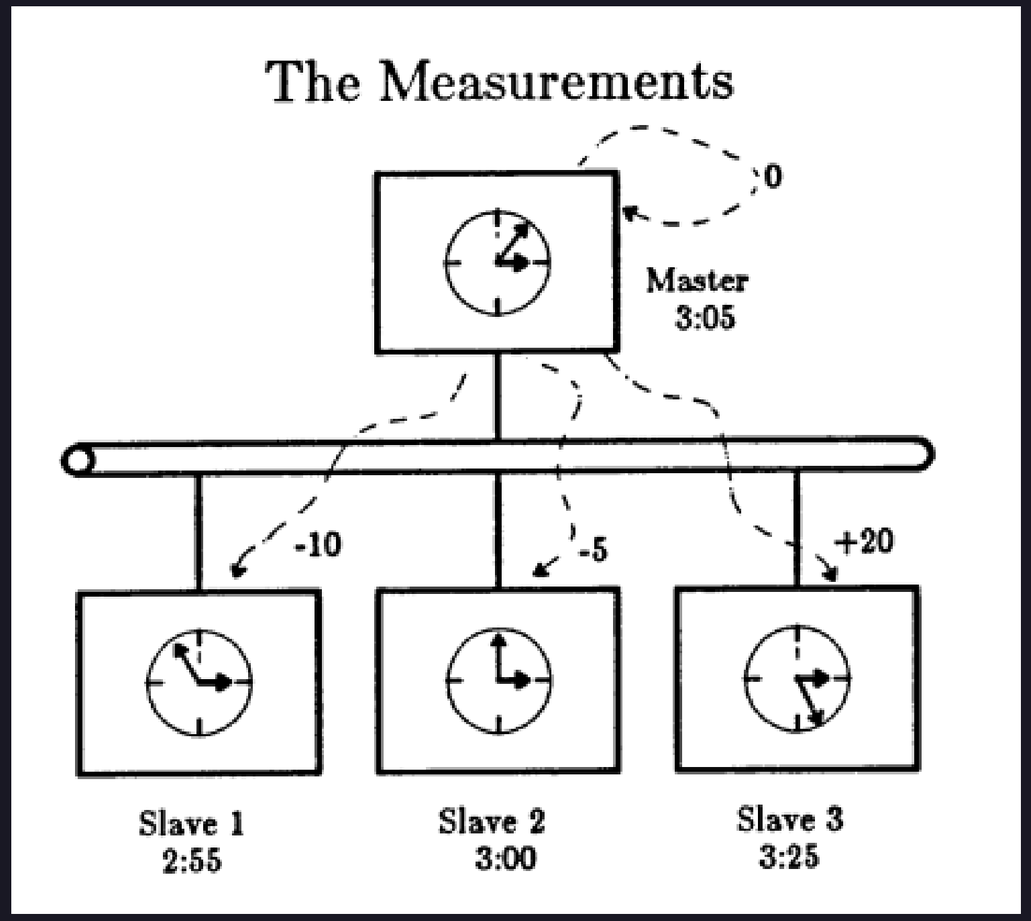
Gusella and Zatti's "Berkeley" Clock Sync Algorithm

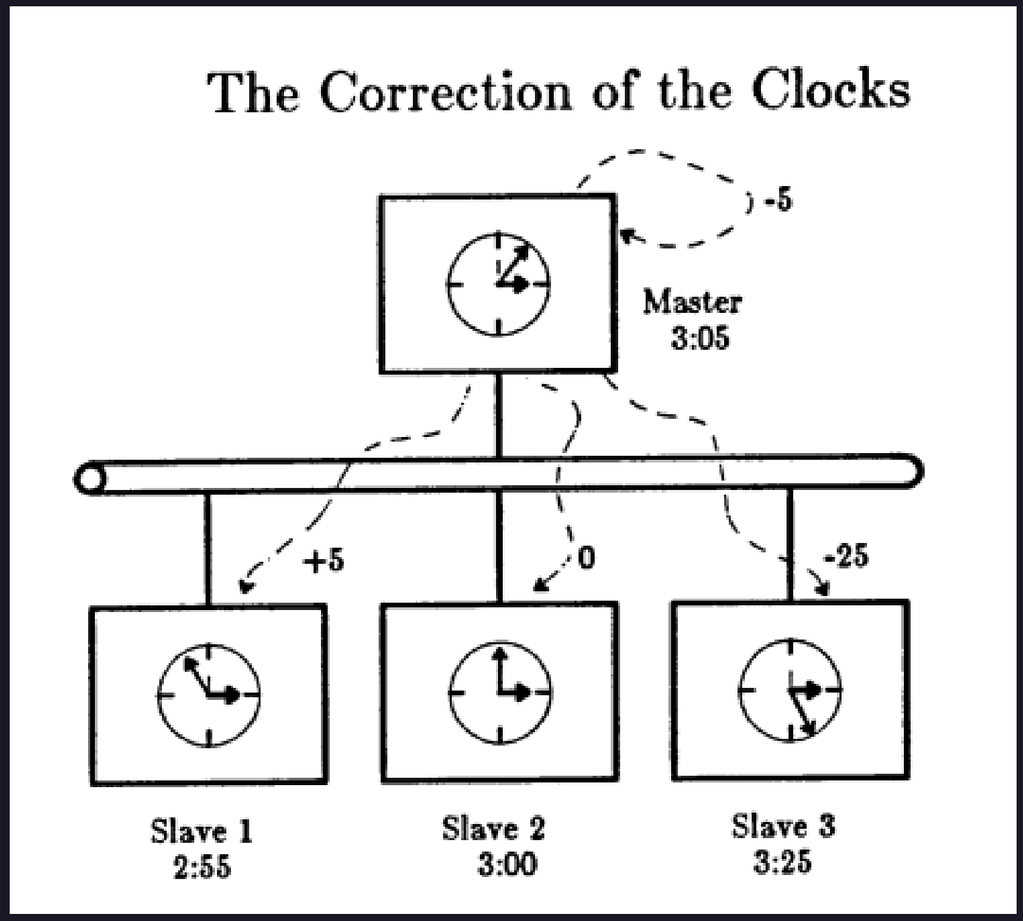
- Synchronizes clocks in a distributed system
- Uses a client-server architecture where one node is the "Master" (aka "Leader") and the others are "Slaves" (aka "Followers")

Gusella and Zatti's Election Algorithm

- Creates *failure transparency* by allowing nodes in the distributed system to *elect* amongst themselves a leader if there's a node failure
- Uses UDP protocol with acknowledgements for increased performance

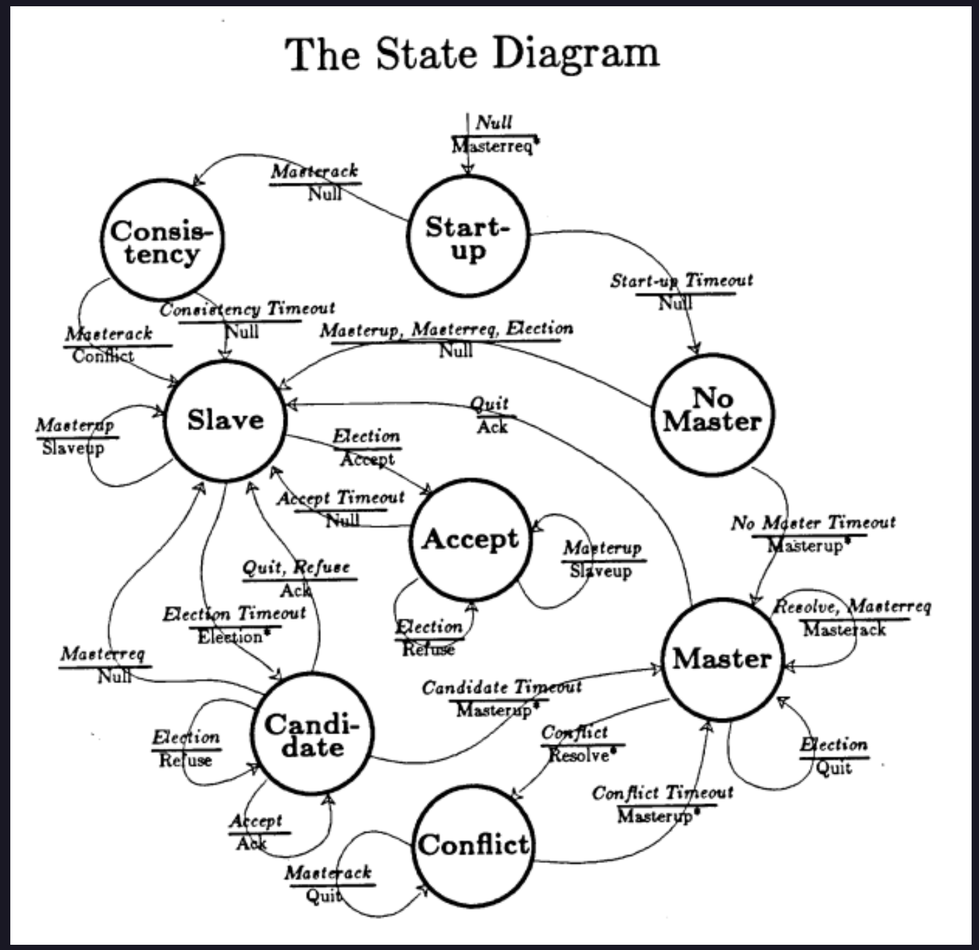
BERKELEY ALGORITHM





Note. **Taken From Gusella and Zatti (1987)**

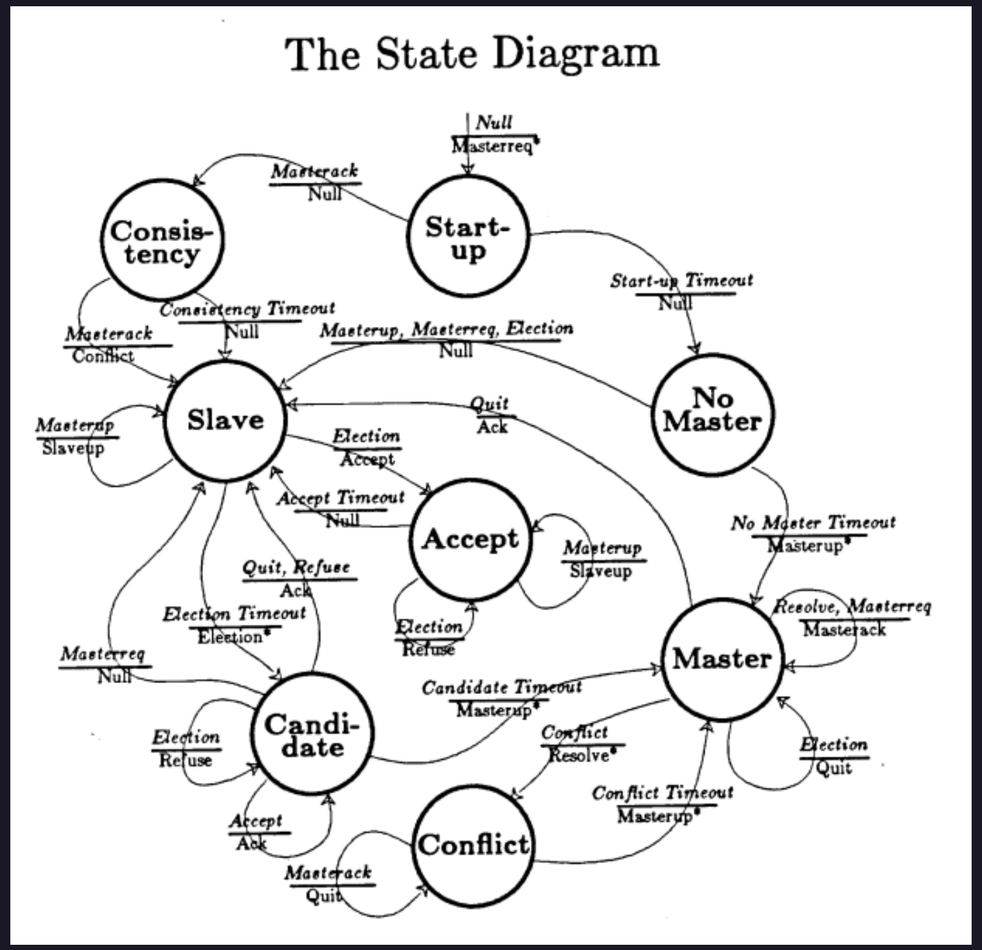
ELECTION ALGORITHM



- Each node may be in one state at a time
- The nodes move between states upon the expiration of a timer or the receipt of a message
- Most of the time, all but one node will be in the "Slave" (aka "Follower") state, while one node will be in the "Master" (aka "Leader")
- At start, a node will broadcast a message to see if a Leader exists
- If not, proceeds to start process of election
- If two nodes are racing to become Leader, the first will try to tell the other to quit

Note. **Taken From Gusella and Zatti (1985b)**

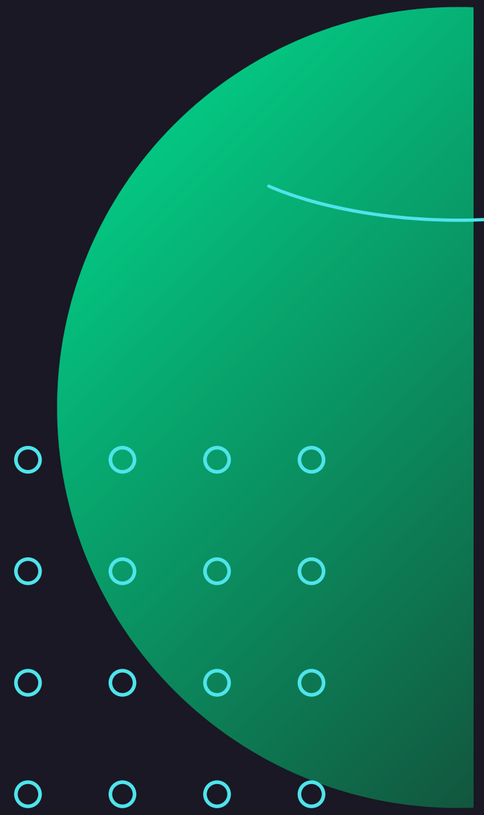
ELECTION ALGORITHM



- In the Candidate state, a node sends Refuse messages to subsequent electees and waits until it no longer hears any Accept messages
- Consistency state is entered after start up to signal a Conflict to a Leader if conflict detected
- Conflict state is entered when a Leader detects a conflict so it can kill the other Leaders

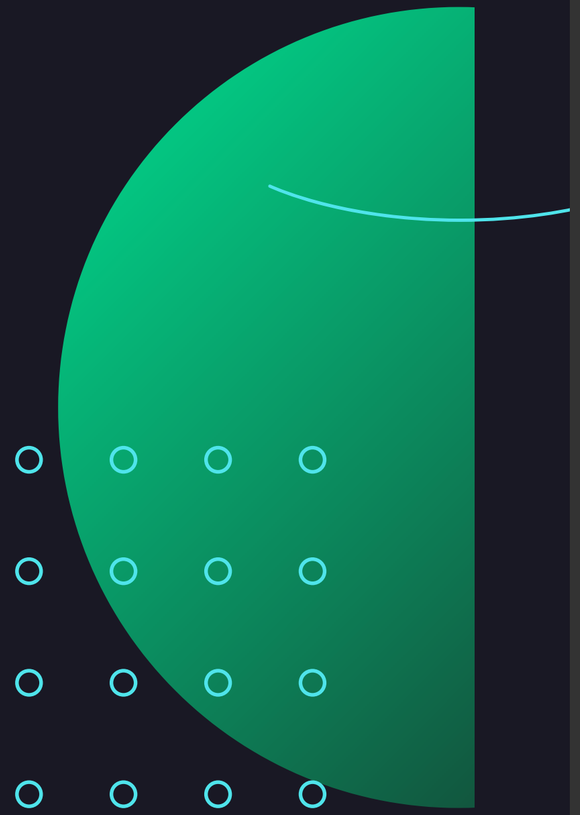
Note. **Taken From Gusella and Zatti (1985b)**

IMPLEMENTATION DETAILS



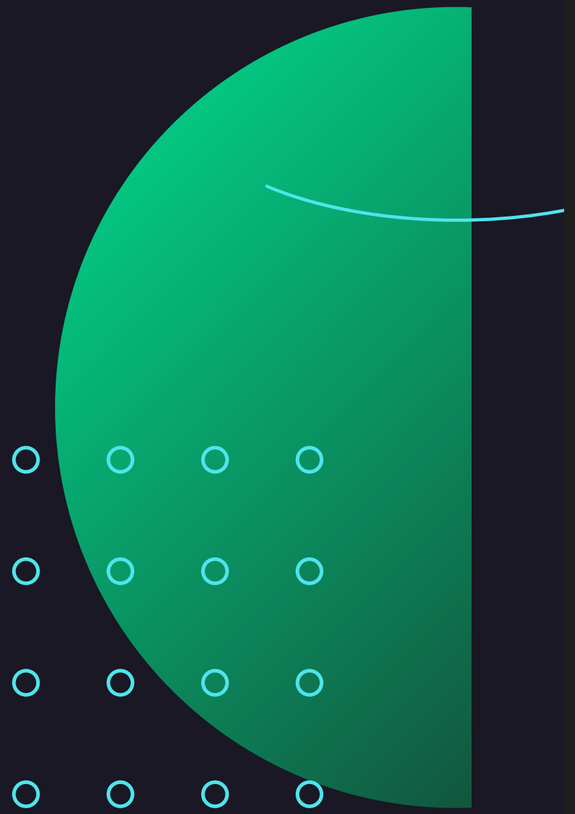
- Docker and docker-compose to start multiple containers, each running an Ubuntu distribution of Linux
- Dockerfile installs during build libfaketime, python, python packages, C build essentials, and some developer tools
- Third-party package called *libfaketime* (wolfcw, 2021) simulates reporting and adjusting clock time and clock drift on the linux machines
- Each node will have the source code installed into /lib/time-daemon
- The time-daemon.py in /lib/time-daemon/src must be started to begin the process of clock correction
- python's built-in *threading* module used to handle blocking logic
- UDP sockets used for all message passing as per Gusella and Zatti's implementation

IMPLEMENTATION DETAILS



```
161     # ==========================================================================
162     def run(self) -> None:
163         while True:
164             print(self.__state)
165 
166             if self.__state == State.START:
167                 self.start()
168             elif self.__state == State.CONSISTENCY:
169                 self.consistency()
170             elif self.__state == State.NOLEADER:
171                 self.no_leader()
172             elif self.__state == State.FOLLOWER:
173                 self.follower()
174             elif self.__state == State.ACCEPT:
175                 self.accept()
176             elif self.__state == State.CANDIDATE:
177                 self.candidate()
178             elif self.__state == State.LEADER:
179                 self.leader()
180             elif self.__state == State.CONFLICT:
181                 self.conflict()
182 
```

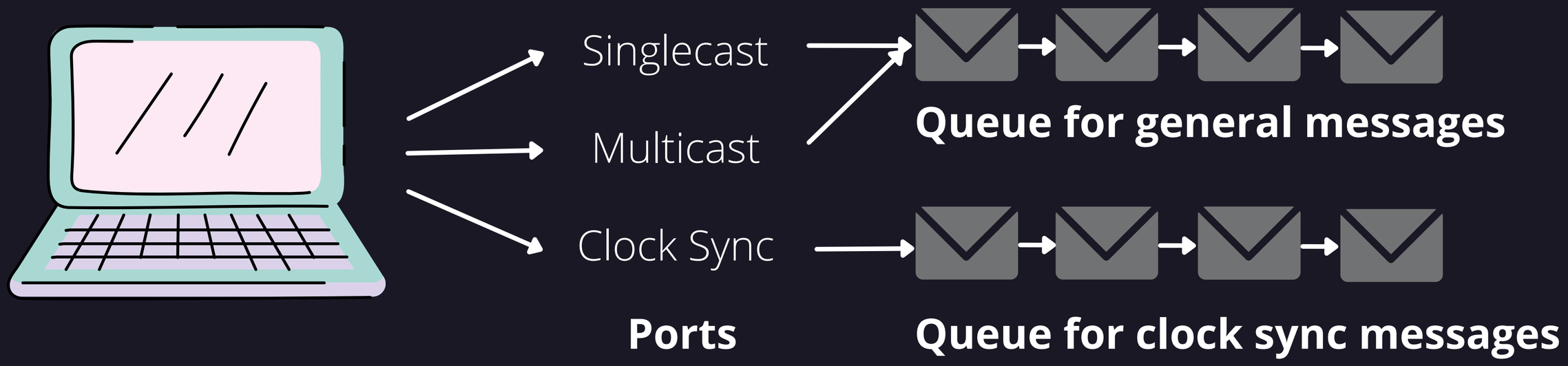
IMPLEMENTATION DETAILS



```
270     # ================================================================================
271     # STATE FUNCTIONS START HERE
272     # ================================================================================
273     def start(self) -> None:
274
275         # Randomly assign a timeout so followers are less likely to elect themselves
276         timeout = random.randrange(RESYNC_RATE, RESYNC_RATE * 2)
277
278         # Then, send out a request for a leader to respond
279         self.send_multicast_signal(Signal.LEADERREQ)
280
281         # Wait here until up to timeout for the response
282         message = self.wait_for_signal_from_queue(
283             self.__message_queue,
284             Signal.LEADERACK, timeout)
285
286         # If the signal was detected, there's a leader
287         if message:
288             self.__first_leader_ip: str = message["ip_address"]
289             self.__state = State.CONSISTENCY
290         else:
291             self.__state = State.NOLEADER
```

IMPLEMENTATION DETAILS

To avoid dropping messages, each node will start a separate thread listening on three different ports. Those messages are deposited into two separate queues and accessed as needed during the logic



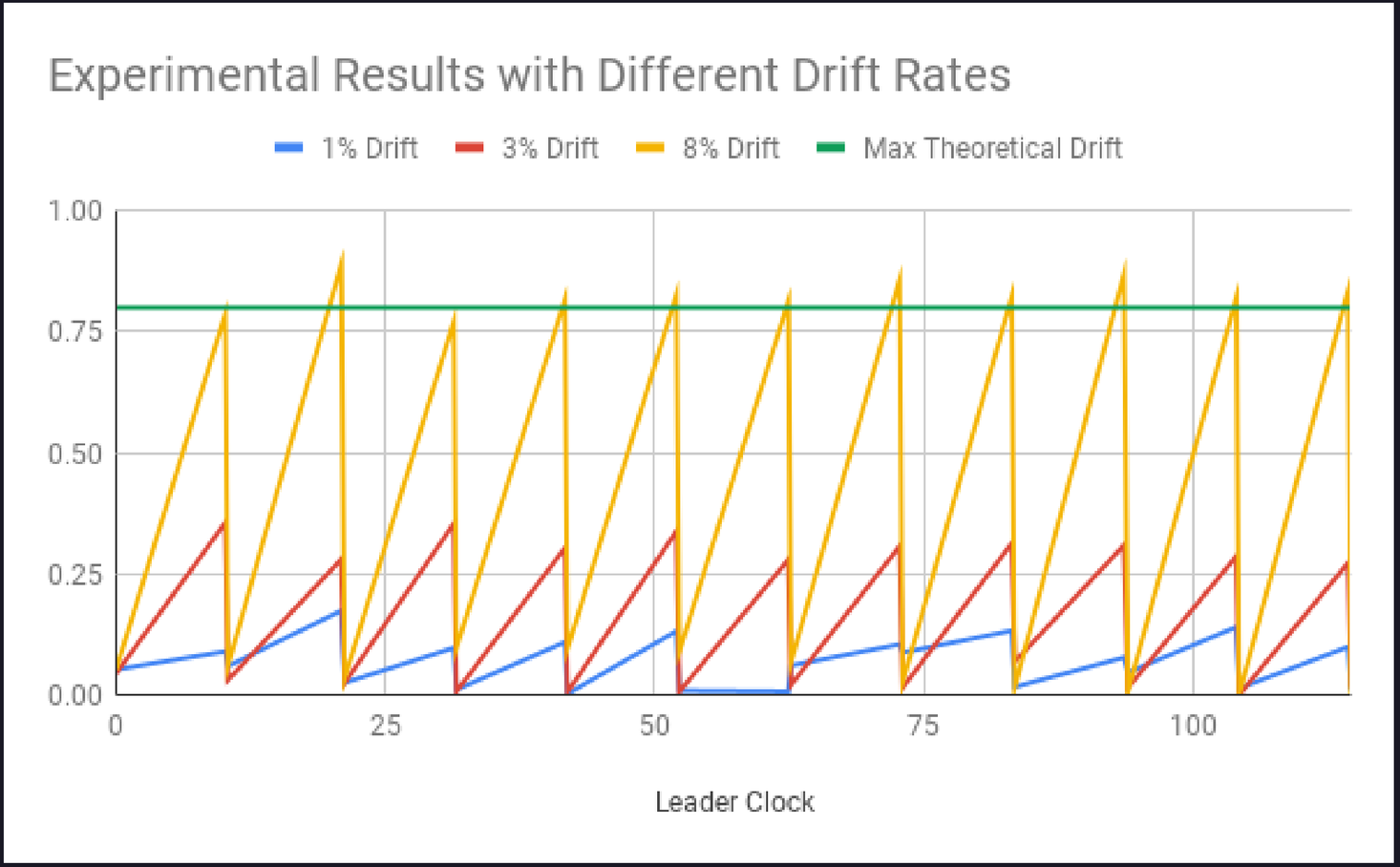
IMPLEMENTATION DETAILS

- The Leader will attempt to resync the clocks every RESYNC_RATE seconds
- Each node will assign itself a MY_CLOCK_DRIFT no more than MAX_CLOCK_DRIFT away from having no drift

```
16  SEC_DECIMAL_ROUND = 2
17  RESYNC_RATE = 10
18  MAX_CLOCK_DRIFT = 0.1
19  MY_CLOCK_DRIFT = round(random.uniform(1-MAX_CLOCK_DRIFT, 1+MAX_CLOCK_DRIFT), SEC_DECIMAL_ROUND)
```

- Based on this, we can calculate a max theoretical drift (with no latency) that we would expect if the system is working correctly

EXPERIMENTAL RESULTS



- Worst clock drift = 8%
- Measurements were recorded just before and just after the Leader fixes all the clocks
- We can see the measured clock drift at the worst for the most broken clock hits just at the maximum theoretical drift, which assumes no latency
- So that's a promising result!

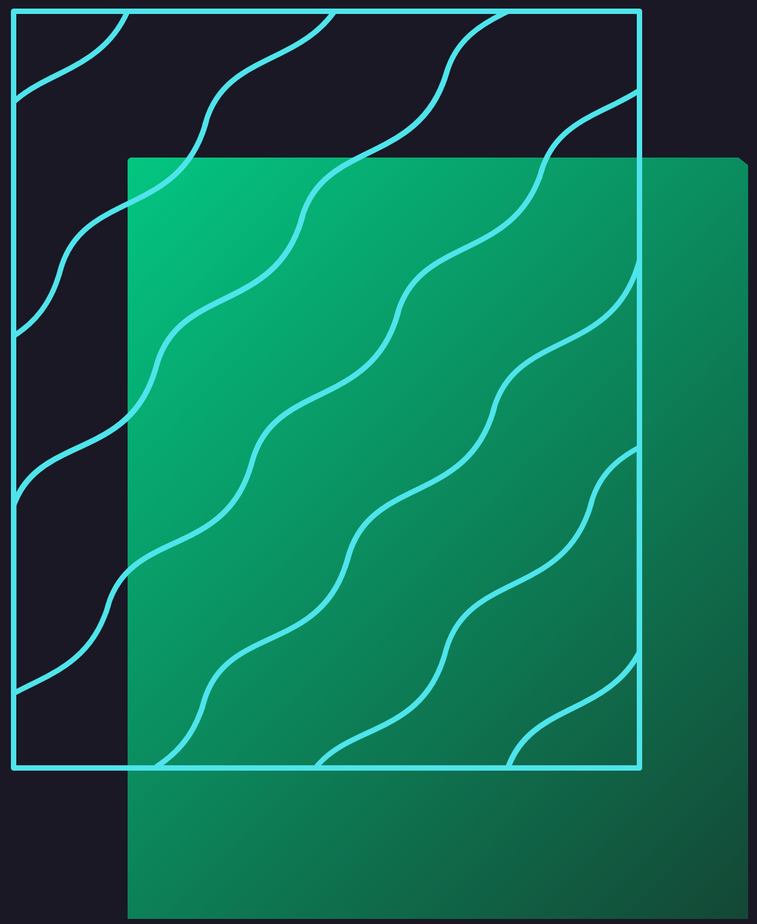
Demo!

DIFFICULTIES

- Testing the correctness of the election algorithm is challenging, since there are many possible states and because of multi-threading
- Probably need to take into account network latency to check correctness
- Since I used libfaketime to simulate clock drift, it was difficult to make sure my time measurements were accurate

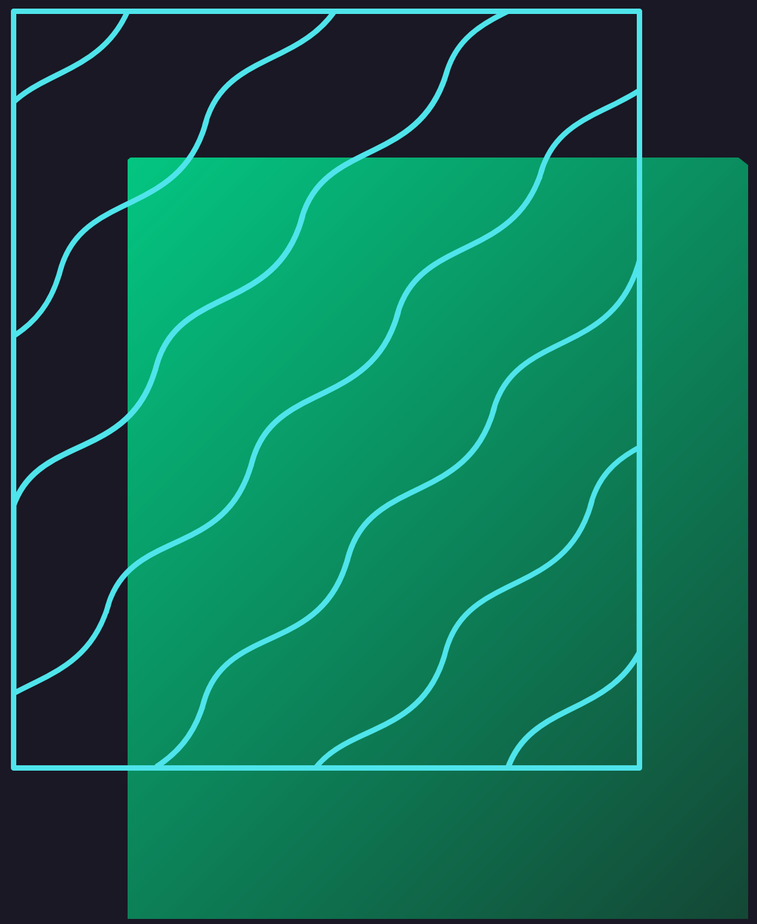
FUTURE WORK & IMPROVEMENTS

- Do more robust testing of the correctness of the election algorithm
- Measure the CPU utilization of this daemon and see what the literature says about how to compare the performance of this algorithm in terms of CPU utilization
- Modify the time daemon to actually directly update the clock of each machine rather than relying on libfaketime



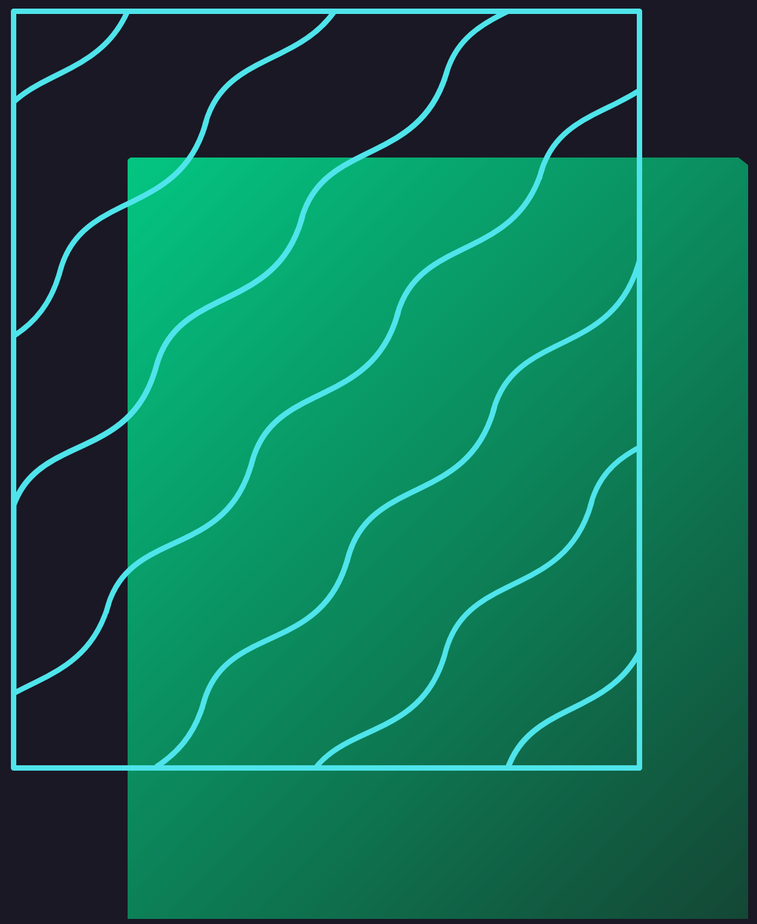
RELATED WORK

- Gusella and Zatti's original paper on the TEMPO Network Time Controller at Berkeley (1983)
- Gusella and Zatti's corresponding Time Synchronization Protocol (1985a)
  - I wrote my own protocol for my implementation
- Gusella and Zatti's corresponding Election Algorithm paper (1985b)
- Gusella and Zatti's paper on the accuracy of TEMPO implemenation at Berkeley (1987)



RELATED WORK

- There are many different types of clock synchronization approaches (Tanenbaum, 2007).
- Mills, D. L. (1991) designer of the Network Time Protocol used in the Internet. It designates "strata" of more accurate machines to allow those of lower strata to get clock sync info from higher strata. Known to achieve worldwide clock accuracy between 1 - 50ms (Tanenbaum, 2019)
- An early client-server approach by Flaviu Cristian (1989) proposes that each client contact a single server to ask for the time



RELATED WORK

- Lamport, L. (1978) proposes a concept of a "logical clock" which is about maintaining the correct order of events in nodes of a distributed system rather than keeping clocks in sync
- Elson et al. (2002) propose another algorithm called Referent Broadcast System that, like the Berkeley algorithm, keeps all nodes in a system in sync with each other

REFERENCES

- Cristian, F. (1989). Probabilistic clock synchronization. *Distributed computing, 3(3)*, 146-158.
- Elson, J., Girod, L., & Estrin, D. (2002). Fine-grained network time synchronization using reference broadcasts. *ACM SIGOPS Operating Systems Review, 36(SI)*, 147-163
- Gusella, R., & Zatti, S. (1983). Tempo: A network time controller for a distributed berkeley unix system. *Computer Science Division,* University of California
- ---. (1985a). The berkeley unix 4.3 bsd time synchronization protocol. *CALIFORNIA UNIV BERKELEY COMPUTER SCIENCE DIV*
- ---. (1985b). An election algorithm for a distributed clock synchronization program. *CALIFORNIA UNIV BERKELEY DEPT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCES*
- ---.(1987). The accuracy of the clock synchronization achieved by tempo in Berkeley Unix 4.3 BSD. *CALIFORNIA UNIV BERKELEY COMPUTER SYSTEMS RESEARCH GROUP*
- Mills, D. L. (1991). Internet time synchronization: the network time protocol. *IEEE Transactions on communications, 39(10),* 1482-1493.
- Tanenbaum, A. S., Steen, M. van, & Steen, M. van. (2007). *Distributed systems : principles and paradigms (2nd ed.).* Pearson Prentice Hall
- wolfcw (2021). LibFakeTime, GitHub repository, https://github.com/wolfcw/libfaketime

Questions?