

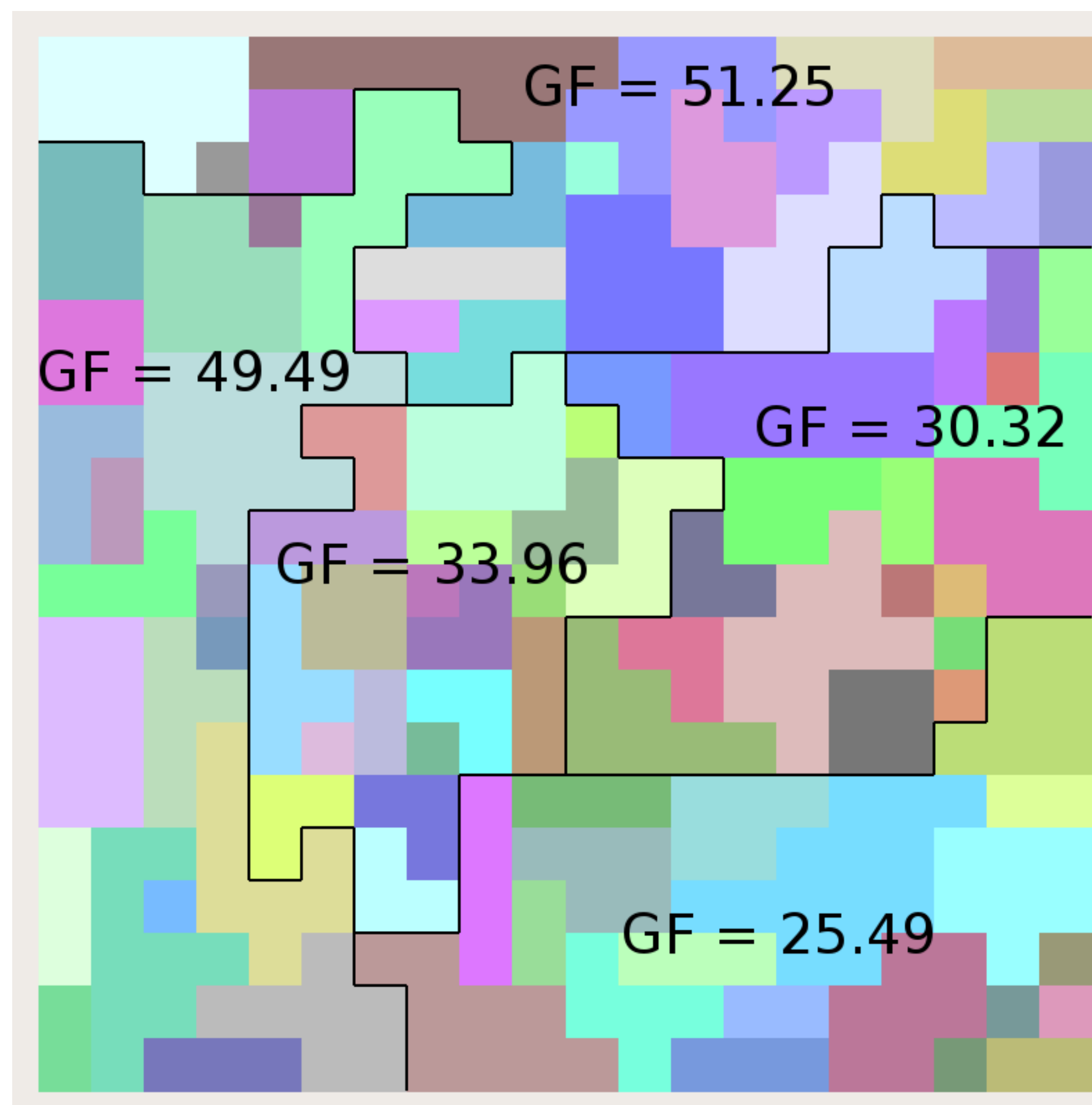
Redistricting Utopia: Using Distributed Computing for Fair Redistricting

Summary of the Original Research

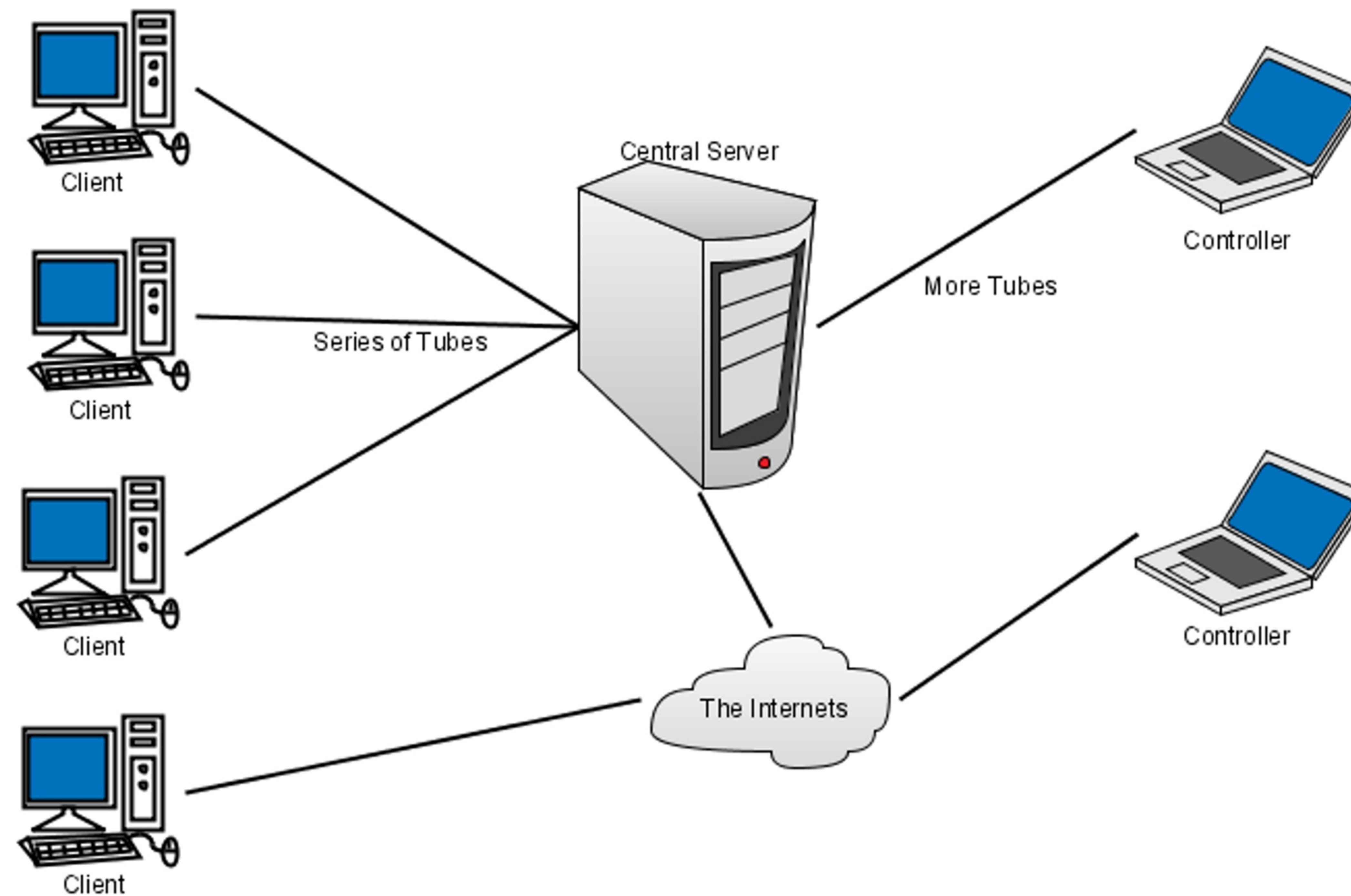
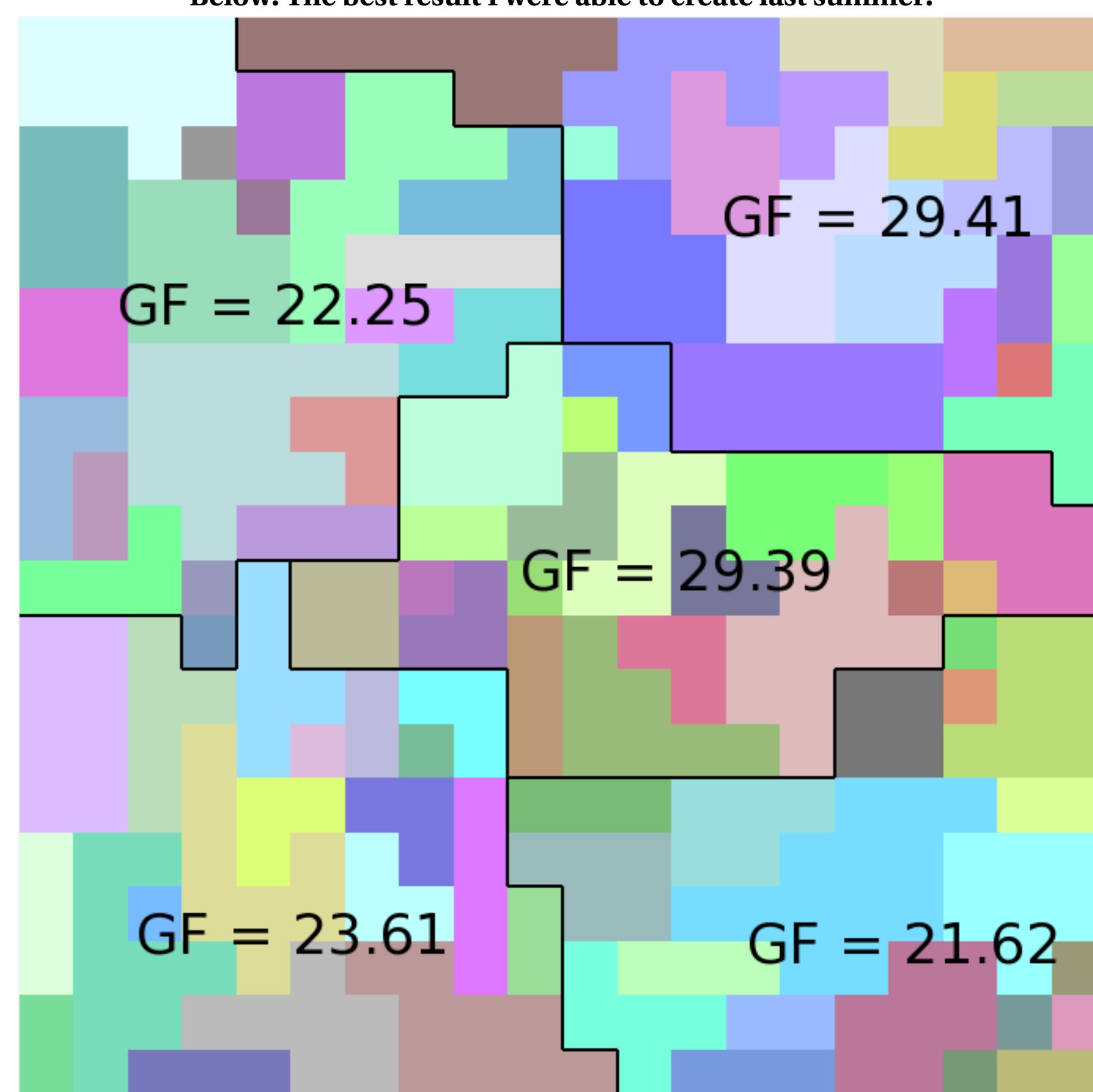
Last summer, I built a program based on a genetic algorithm that proved to be very effective at finding fair redistricting plans for synthetically generated sets of precincts. However, it only ran on one computer and was quite slow, often taking 2 days to find an acceptably good solution. The genetic algorithm works by creating random strings of 1s and 0s (bit strings) that represent a single solution to the problem and are analogous to DNA. These solutions are given a fitness rating based on how good of a solution it is. These solutions and, specifically, their bit strings breed with each other to create child solutions. Better solutions are more likely to reproduce, so the overall fitness of the population of solutions increases over time.

Comparisons with our Previous Work

Last summer, my program was able find good solutions to synthetically generated maps and this summer my program improved dramatically. By using distributed computing and reimplementing the algorithm to ensure its correctness and improve its efficiency. While the result shown below is not quite as good as the result from last summer, the example result from this summer was found in a few hours instead of a few days. Furthermore, the GF rating of the districts is not the whole story and I suspect that the apportionment in this summer's example is better than last summer's due to the weights I used.



Above: A sample result from my program this summer.
Below: The best result I were able to create last summer.



An abstract diagram of the structure of my distributed program.

Distributed/Cluster Computing

I made use of distributed computing to increase the power of our system. Last summer, my program ran on one computer at a time. This summer, I reimplemented the program and built networking in from the beginning. Like last summer, the program used genetic algorithms which are based on evolution. Extending that analogy, each computer is its own island on which isolated evolution takes place. Occasionally, one animal swims from one island to another so that they can all benefit from each other's progress while continuing to evolve along their own evolutionary paths.

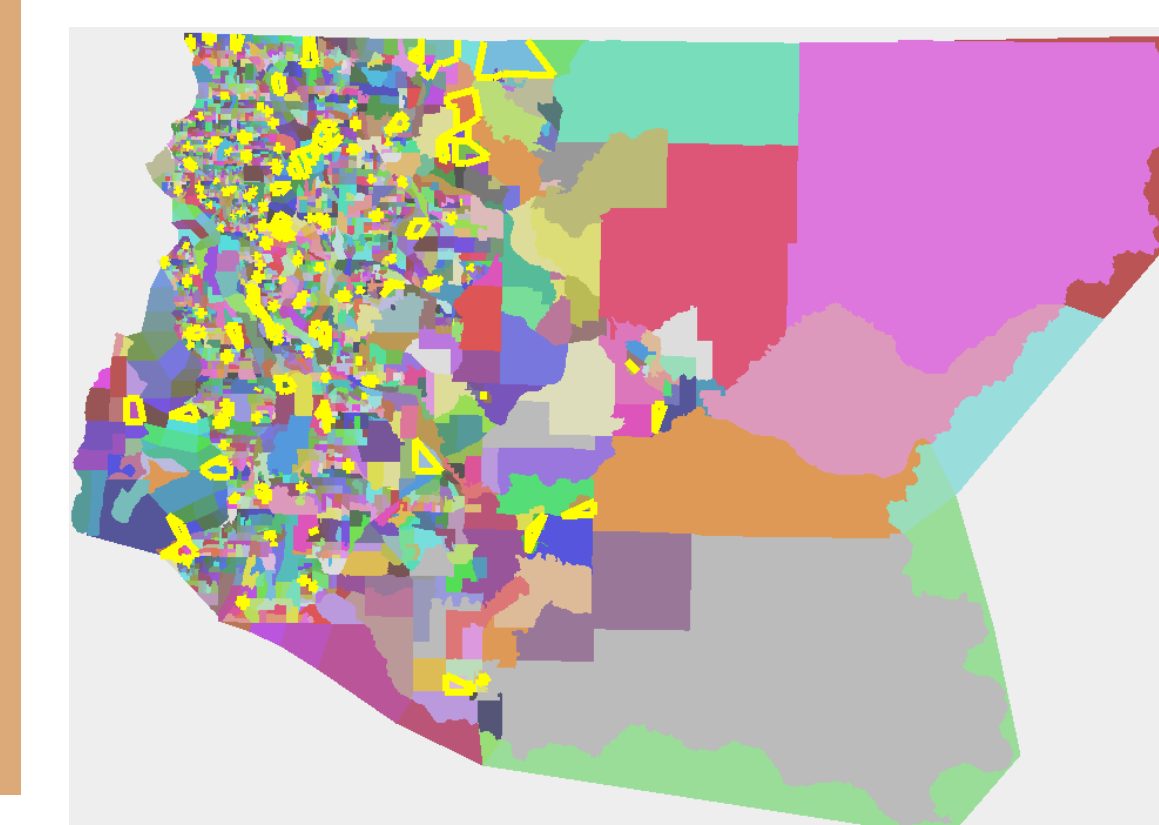
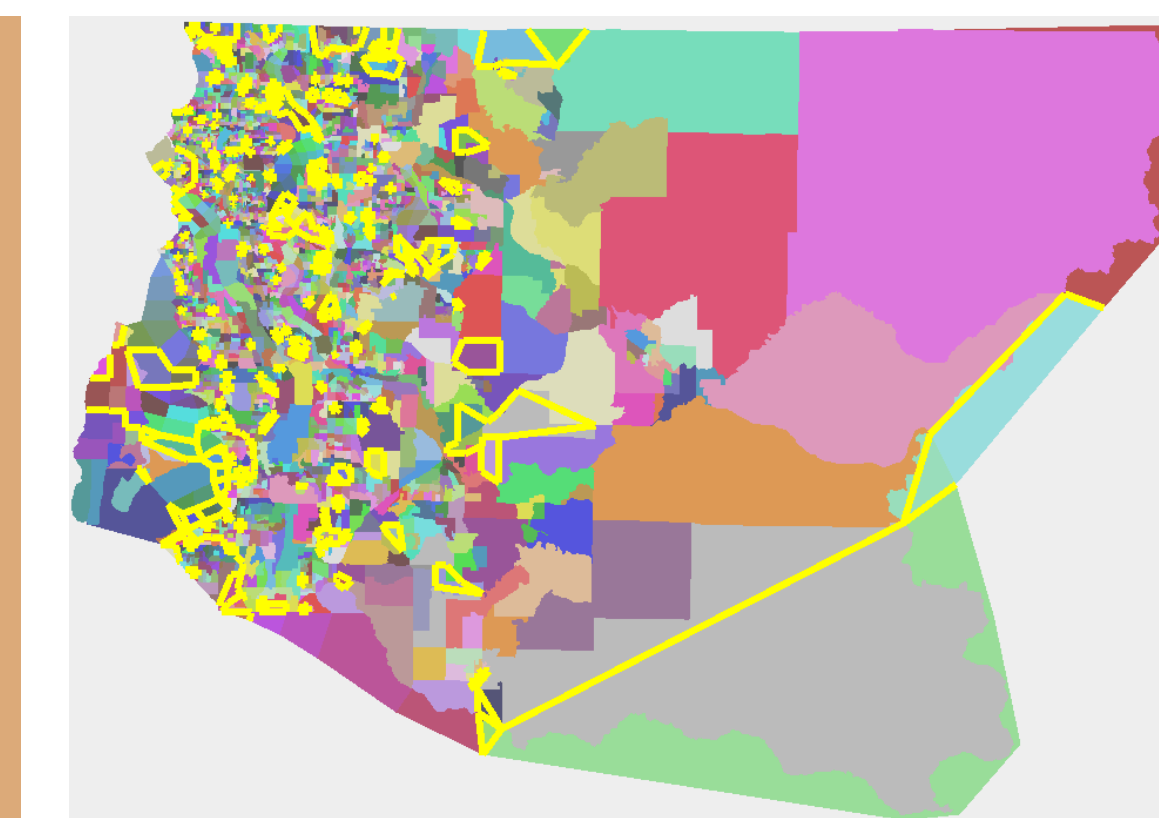
There were several reasons for reimplementing our algorithm: there were some lingering and troublesome bugs in the code from last summer, it allowed me to build networking in at a more useful level, and write it in a higher level language so that I could spend less time writing code and more time analyzing the results. For these and other reasons, I chose to write the program in Python. Unfortunately, Python did not provide as many benefits as I had hoped. As it turned out, most of the summer was spent focusing on writing robust and reliable networking code that could respond to errors correctly (see The Protocol, A Technical Aside).

Alternate approaches could have been taken, such as using one of the distributed object modules that allows a single program to run across multiple computers in an almost transparent way. However, I preferred the flexibility of each computer running its own simulation that could run using different parameters and still share solutions with other computers.

This Summer's Results

This summer I began using a data set representing the precincts in King County. This increased the number of precincts from the 100 I was using last summer to 2754 (see Hurdles to Using the King County Data for more on why that many). Not surprisingly, such a drastic increase in the size of the data set significantly increased the amount of time necessary to find a good solution not only because it takes longer to calculate the fitness of a larger genome, but also because the small changes that happen at each generation have less of an impact on the larger genomes. Pictured to the right are two images of solutions my program found over the course one run this summer. The top one was from early in the run and has roughly 240 districts (far from the desired 7 districts). The bottom image has 159 districts which shows significant improvement, although there is still clearly a long way to go.

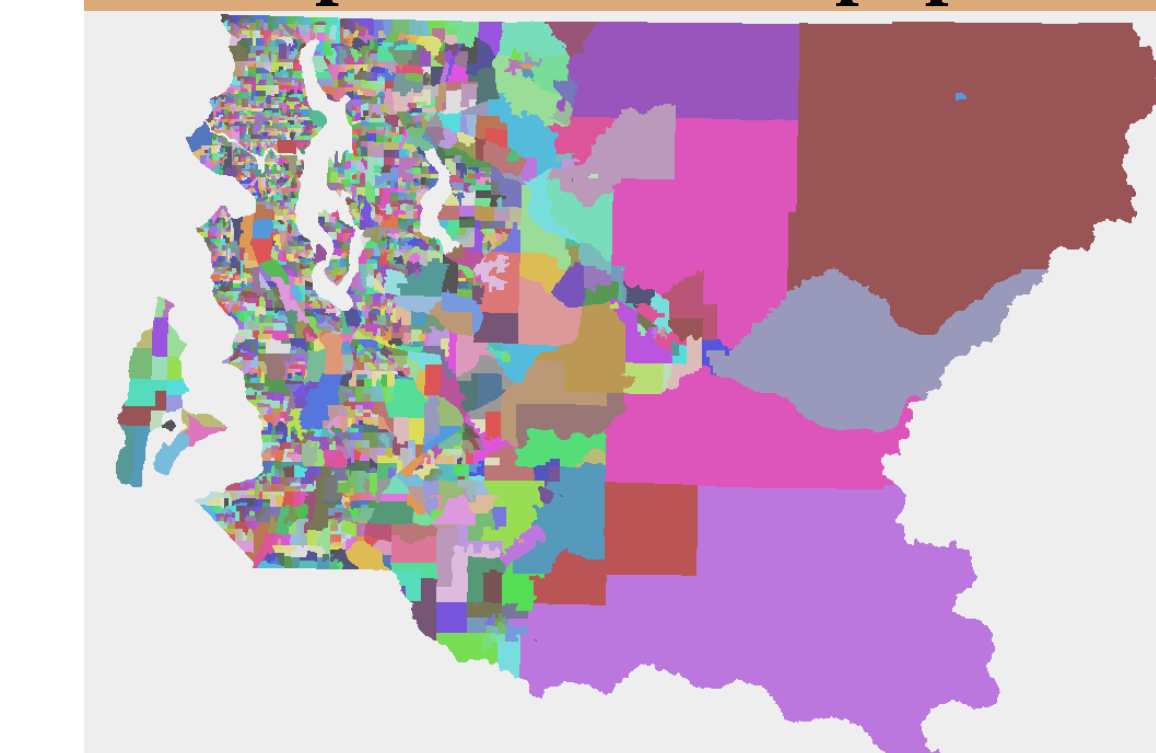
The hope is that many clients running for several weeks can find a good solution to the King County data. Further work will also include trying different parameters to see what settings provide the best results. Therefore, my research will continue in the hopes of improving the program and providing better results. It will also be essential to expand the number of computers my program is running on to increase the overall computing power of the cluster. I believe that with a little more time and effort, my program will be able to come up with very strong solutions.



Hurdles to Using the King County Data

Last summer, I spent a significant amount of time trying to write a general purpose program that could fill in the empty spaces between precincts caused by lakes, rivers, and some errors in the GIS data. Such a general purpose solution was difficult to create and, in the end, created more problems than it solved. It also did not solve other problems such as precincts with multiple, unconnected parts and precincts that are wholly within other precincts.

So this summer, I decided that I would rather have at least one set of GIS data that my program could process than have a partially working general purpose solution. Prof. Bentson offered to take on this project and spent 2 months of the summer working to get the King County data into a format my program could use. Most of the work he did was adding precincts to manually fill in the bodies of water, connect Vashon island to the mainland, and fix the problems caused by previously mentioned multi-part, disjoint precincts and precincts embedded inside other precincts. Filling in the bodies of water and connecting Vashon to the mainland added 207 synthetic precincts while handling embedded and multi-part precincts caused 114 precincts to be merged down to 46. This created a net increase of 139 precincts or about 5%. Adding all these precincts greatly increased the number of precincts in King County, but that does not affect the results because all the added precincts have populations of 0.



A map of King County precincts without the added filler and merged precincts (see Hurdles). You can see the precincts Prof. Bentson added by comparing this picture to the ones below.

The Protocol, A Technical Aside

I designed the network protocol to be as hands off as possible, only sending messages when necessary. It was also important to design the protocol so that the clients would be very autonomous and could spend most of their time finding better solutions instead of processing requests from the server. These requirements drove me to create a heavily threaded, stateless, and asynchronous (either side can send data without the expectation of a response) protocol.

The client and controller each have separate threads for sending and receiving data while the server maintains a send and receive thread for each client and controller connected to the server. This was necessary because the large amounts of data being sent back and forth sometimes saturated the send and receive buffers on one or both ends of the connection. It also allowed for greater efficiency as the networking threads only run when needed and will mostly sit in the background while the client searches for better and better solutions. In layman's terms, I made sure different parts of program could effectively run simultaneously. This prevented problems caused by the send and receive portions of the program waiting on each other. If they both run at the same time, then a slow down in one part will not cause problems for the other parts of the program.

Despite building the protocol on top of TCP, I did not want any part of the program to assume it knew what was coming next (which is often a fair assumption given the reliable delivery that TCP can provide). Therefore, my protocol is stateless and the message packets were designed to contain enough information that the receiver could always respond correctly. Put more simply, the program has no memory, it will wait until it gets a message, respond to the message, and then go back to waiting with no memory of what it has done or sent before. This is not a problem because the networking protocol primarily exists to make sure the clients are running the correct version of code and send genomes back and forth between the server and the clients.