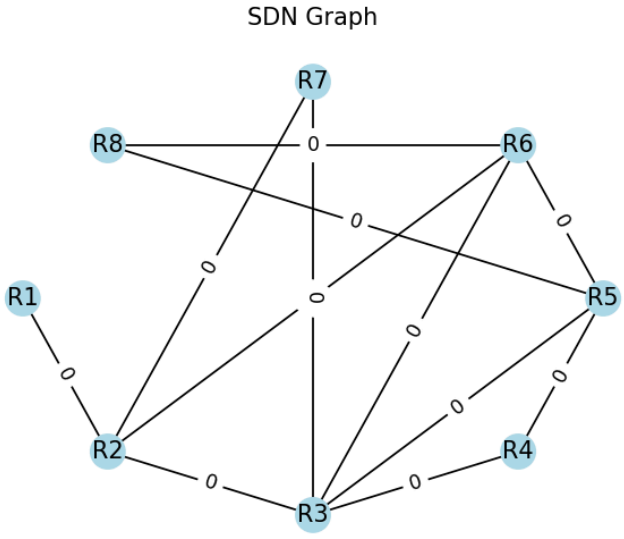
Jake Mayer

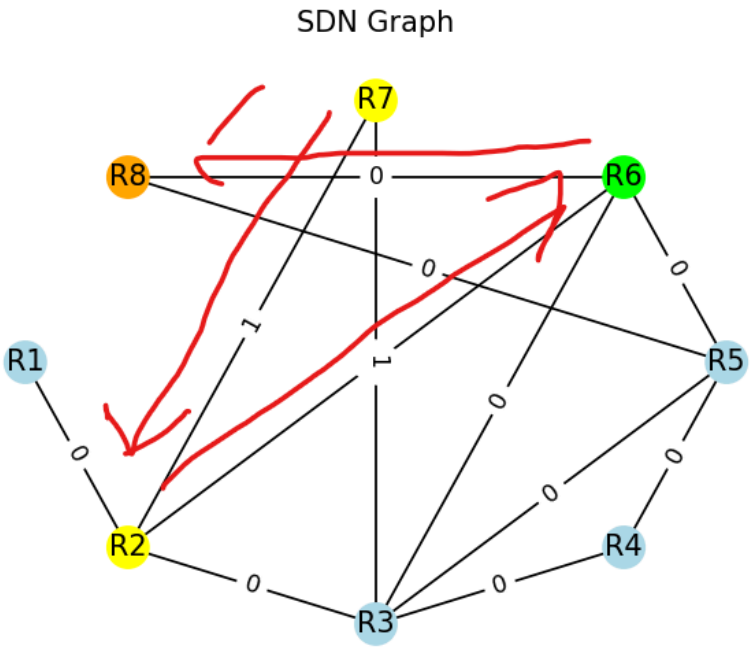
CSC4501 Assignment 4 Question 4 Design Document

In designing this SDN controller I decided to use object oriented design for it, and to give it a bunch of functions for it to run like a real SDN controller. The SDN has functions designed for creating flowtables, updating flowtables, creating a network graph, managing nodes on that network graph, managing the links between nodes on the graph, and sending flows using flowtables and visualizing it with the network graph. In the creation of the graph, MatplotLib and networkX python libraries gave me many options and definitely allowed me to make everything much better and I even figured out how to sort of animate the flows traveling along the graph. Finding shortest paths were done using Dijkstra's Algorithm which is an algorithm widely used for finding the shortest path between nodes. Flowtables follow the design below to the right, giving the destination and the "next hop" on the path to the destination. Here is an idea of what the graphs usually look like. This one was produced directly by the program. It marks usage rates of each link between the nodes and then those increases are incorporated into Dijkstra's algorithm to do a kind of load balancing.



R6 Routing Table	
Destination	Next Hop
Destination	Next Hop
R3	R3
R3	R3
R1	R1
R3	R3
R1	R1
R1	R1
R5	R3
R8	R1
R2	R1
R7	R1
R4	R1

Here is a snapshot of what the graph looks like during a flow of R7 to R8. From this document the animation isn't viewable so the path is drawn the graph, which the flow did take the shortest path.

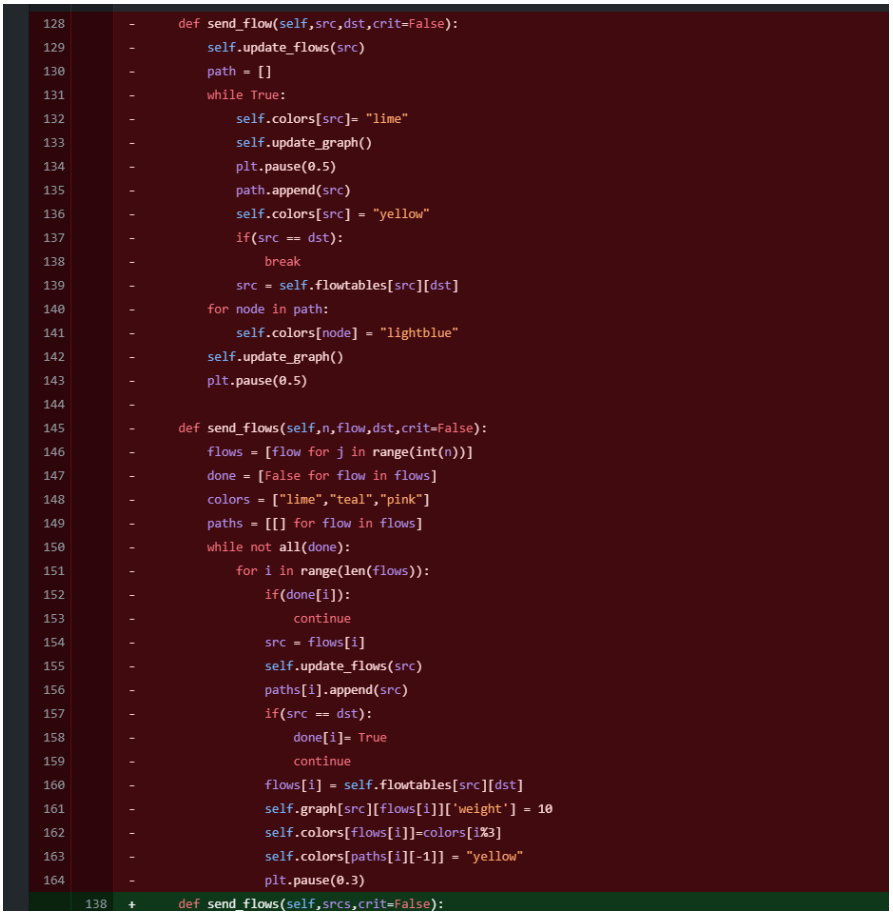


Finally design-wise is the Command-Line Interface (CLI). Commands are intended to be very simple but offer a lot for the user to mess around with, in an attempt to emulate a real control plane where customization is very prevalent. Most commands were designed to be capable of using decent amounts of args. For example, when using "route" you can even send over 10 flows consecutively.

```
welcome to the SDN Control Plane!
>> route 3 7 8
>> help

List of Commands: (Use numbers to represent nodes!)

  node x                - Creates x number of nodes (e.g., 'node 5').
  link x1 x2 [x3...]    - Links node x1 to x2, x2 to x3, etc.
  del x1 [x2...]        - Deletes nodes x1, x2, etc.
  del link x1 x2 [x3...] - Deletes link(s) between x1 and x2, x2 and x3, and so on.
  route x1 x2 [x3...]   - Sends flow from x1 to x2 and x2 to x3 and so on.
  table x               - Displays the routing table for node x.
  q                     - Exits the program.

>>
```

Uniqueness Verification

1. A major issue I really struggled with was getting the flows to work properly with the flowtables. I tried various ways of coding and re-coding it, but I finally figured out how to set up a for loop for it properly. At some point my code was trying out separate functions to try it in different ways. (The red indicates this got scrapped).



```
-     def send_flow(self,src,dst,crit=False):
-         self.update_flows(src)
-         path = []
-         while True:
-             self.colors[src]= "lime"
-             self.update_graph()
-             plt.pause(0.5)
-             path.append(src)
-             self.colors[src] = "yellow"
-             if(src == dst):
-                 break
-             src = self.flowtables[src][dst]
-         for node in path:
-             self.colors[node] = "lightblue"
-         self.update_graph()
-         plt.pause(0.5)
-
-     def send_flows(self,n,flow,dst,crit=False):
-         flows = [flow for j in range(int(n))]
-         done = [False for flow in flows]
-         colors = ["lime","teal","pink"]
-         paths = [[] for flow in flows]
-         while not all(done):
-             for i in range(len(flows)):
-                 if(done[i]):
-                     continue
-                 src = flows[i]
-                 self.update_flows(src)
-                 paths[i].append(src)
-                 if(src == dst):
-                     done[i]= True
-                     continue
-                 flows[i] = self.flowtables[src][dst]
-                 self.graph[src][flows[i]]['weight'] = 10
-                 self.colors[flows[i]]=colors[i%3]
-                 self.colors[paths[i][-1]] = "yellow"
-                 plt.pause(0.3)
+     def send_flows(self,srcs,crit=False):
```

The end result I am very happy with, but it is pretty difficult to fit it all in here. I will leave a image of it on the next page, sorry for the lower quality.

```python
#function for sending flows on the graph
def send_flows(self,nums,crit=False):
    srcs = []

    #loop through every arg item
    for n in nums:

        #validate inputs
        try:
            temp = int(n)
            if not self.graph.has_node("R"+n):
                print("Missing nodes.")
                return
            srcs.append("R" + n)
        except ValueError:
            print("Please use only numbers.")
            return

    #begin sending flows
    for i in range(len(srcs)-1):
        src = srcs[i]
        dst = srcs[i+1]
        #update for latest paths
        self.update_flows(src)
        #go to next flow if no path exists
        if self.flowtables[src][dst] == {}:
            print(f"No path from {src} to {dst}.")
            continue
        path = []

        #set color of goal
        self.colors[dst] = "orange"

        #loop through flowtables until the end is reached
        while True:

            #change color of current node
            self.colors[src]= "lime"

            #update graph to make flow animated
            self.update_graph()
            plt.pause(0.5)

            #add node to path traversed
            path.append(src)

            #change color of previous node
            self.colors[src] = "yellow"

            #leave loop if destination is reached
            if(src == dst):
                break
            #adjust to next node using flowtables
            src = self.flowtables[src][dst]

            #increment usage value of link between nodes (influences load balancing as well)
            self.graph[path[-1]][src]["weight"]+=1

        #reset node colors
        for node in path:
            self.colors[node] = "lightblue"

        #update graph
        self.update_graph()
        plt.pause(0.5)
```

2. Here is my hash watermark (its at the top of my sdn.py)

```
1  # HASH WATERMARK: f4442691a8dcb41258233448d5429f4064778242e0d75f4f92e9ef464c3263ae
2  # MADE BY JAKE MAYER
3
4
```