Design Implementation

1. Introduction

This document explains a simple SDN controller written in Python. The controller keeps track of the network layout and decides how to send data. It can handle lost links and balance traffic in different ways.

2. Architecture Overview

The code is split into parts for clarity. One part manages the network diagram. Another part finds the best routes for data. A third part keeps track of active data flows. There is a text interface for adding nodes and links and starting or stopping traffic. A final part draws the network picture and shows how much each link is used.

3. Routing Decisions

To find the main route the controller uses a method that picks the fastest path. To find a backup route it tries removing each link along the main path and finds the next best option. For normal traffic it switches back and forth between main and backup paths to balance the load. Important traffic runs on the backup path only if the main path fails. Traffic also has a priority number and high priority data is set up first.

4. Command Line Interface

The user types text commands to control the network. Commands let you add or remove nodes and links. Other commands let you start or stop data flows. There is a command to simulate a broken link and one to check the route for any flow. You can also view all flow tables or save a picture of the network.

5. Visualization

The visual module draws the network as a set of circles and lines. Line thickness shows how much data is passing through each link. It also prints a list of active data flows and how busy each link is.

6. Conclusion

This simple design makes it easy to test key SDN features in one script. You can extend it to use better path finding, add a graphical user interface, or support more traffic rules.

7. Implementation Challenge

When adding the backup path feature, the first attempt changed the network layout by mistake. The code removed links but did not put them back right away. This broke the network for later steps. Here is the broken code:

```
for u, v in primary_edges:
    topo.remove_link(u, v)
    alt = topo.shortest_path(src, dst)
    if alt != primary:
        backup = alt
```

To fix it, we saved each link's weight, removed it temporarily, found the path, and then restored the link before moving on. That way the network stayed correct each time. Here is the fixed code:

```
for u, v in primary_edges:
    w = topo.adj[u][v]    # save weight
    topo.remove_link(u, v)
    alt = topo.shortest_path(src, dst)
    if alt and alt != primary:
        backup = alt
    topo.add_link(u, v, w)  # put link back
```