Statistical Multiplexer Simulation

CPE400 Final Project

Dat Luu
Yuta Miyake

I. Introduction

Statistical multiplexer is a key concept in computer network. Because network is not always in use, link sharing must be handled efficiently during network's' burst. Statistical multiplexer handles link sharing based on statistic demand; slot will be allocated based on priority and need. Understanding the concept, we implemented a program to simulate and visualize how statistical multiplexer handles link sharing based on priority.

II. Contribution

The condition is that each source/destination has different missions, or different types of packet to send to a specific source/destination. Each router has a pre-set hierarchical priority base for handling packet types; each router will provide more buffer and allocate more slot for a specific type of packet. The network's topology is arbitrary at the time of simulation; not all routers are connected to each other and link cost for each router will be different. Moreover, each router only knows its neighbours at the time of transferring the packet.

An optimized path, for packets to traverse through the arbitrary network, will depend on the delay at each router; the link cost for packet to traverse between routers; the number of steps for sending "advertisement" and getting network topology for shortest path. The delay at each router will depend on how the router prioritize the packet being sent to it; more buffer and more slot allocated for a packet type will result in less delay. The link cost will be pre-defined, and we assume there will be none link failure. The number of steps for sending "advertisement" through the network, for realizing network topology and getting shortest path, will increase the delay, and will have a significant impact on determining the shortest path; flooding packets across the network for getting shortest path may result in a greater delay. Figure 1 shows the problem statement of our project, and the network topology we will be working on.

Understanding the core concept, we define the challenges that will be addressed in the project. First, the project will simulate statistical multiplexer method, and output an optimized path for packets flowing between its source and destination; the program will also output the delay, loss, and throughput. Second, the project will account for dynamic changes in how routers prioritize the packet flow type. Finally, both static and dynamic scenarios will be visualized. Figure 2 shows the challenges being addressed by our project.

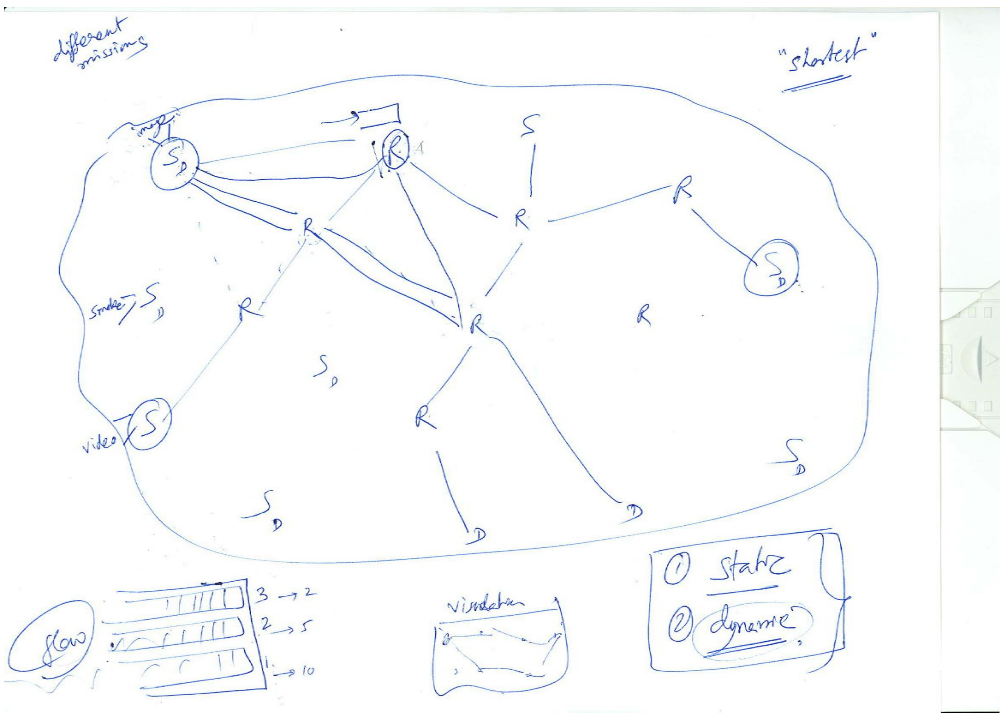
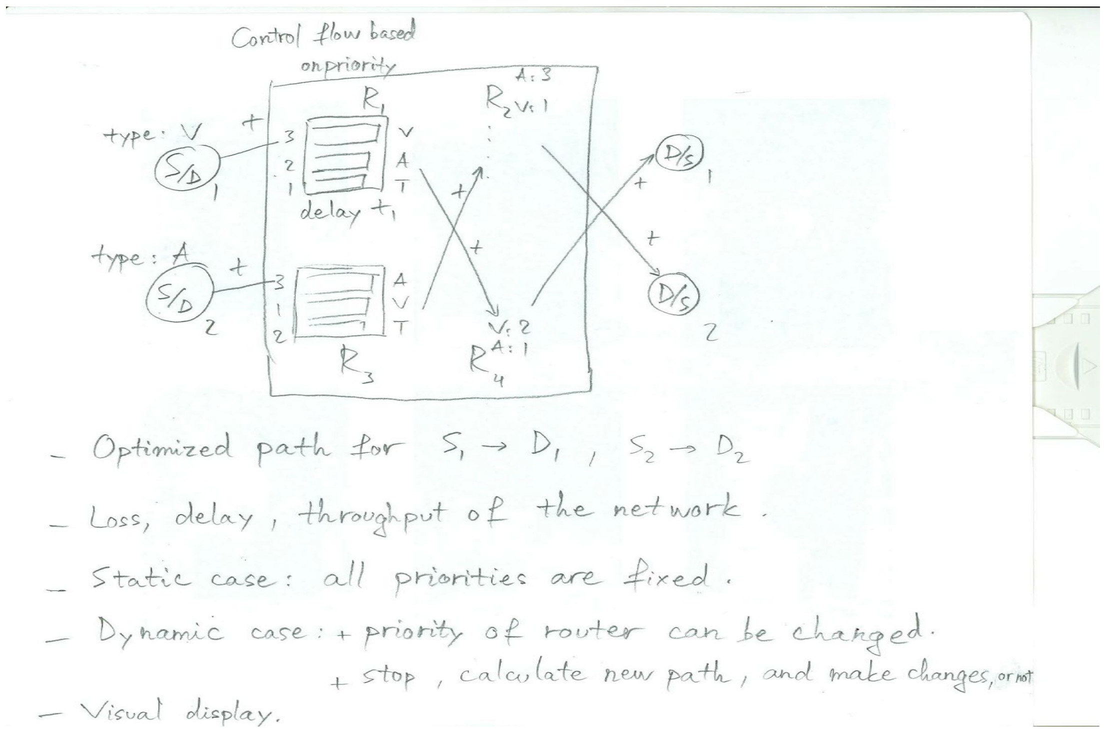


Fig. 1: project problem statement, statistical multiplexer handles link sharing on a network with arbitrary topology; not all routers are connected, and link cost for traversing routers will be different.



- Optimized path for $S_1 \rightarrow D_1$, $S_2 \rightarrow D_2$
- Loss, delay, throughput of the network.
- Static case: all priorities are fixed.
- Dynamic case: + priority of router can be changed.
  + stop, calculate new path, and make changes, or not
- Visual display.

Fig. 2: Challenges that will be addressed.

III. Code Explanation

A. Platform: the program was written in Java program language; it can be compiled and run on both Windows and Linux operating system.

B. Compile and Execute:
   a. Windows: we recommend using the NetBean IDE to compile and test run the code.
   b. Linux: Java JDK is required for compiling the project. Open a terminal window and locate the source code folder. Input the following two lines

```
javac *.java // for compiling the project
java Driver // for test run on the project
```

C. Diagram

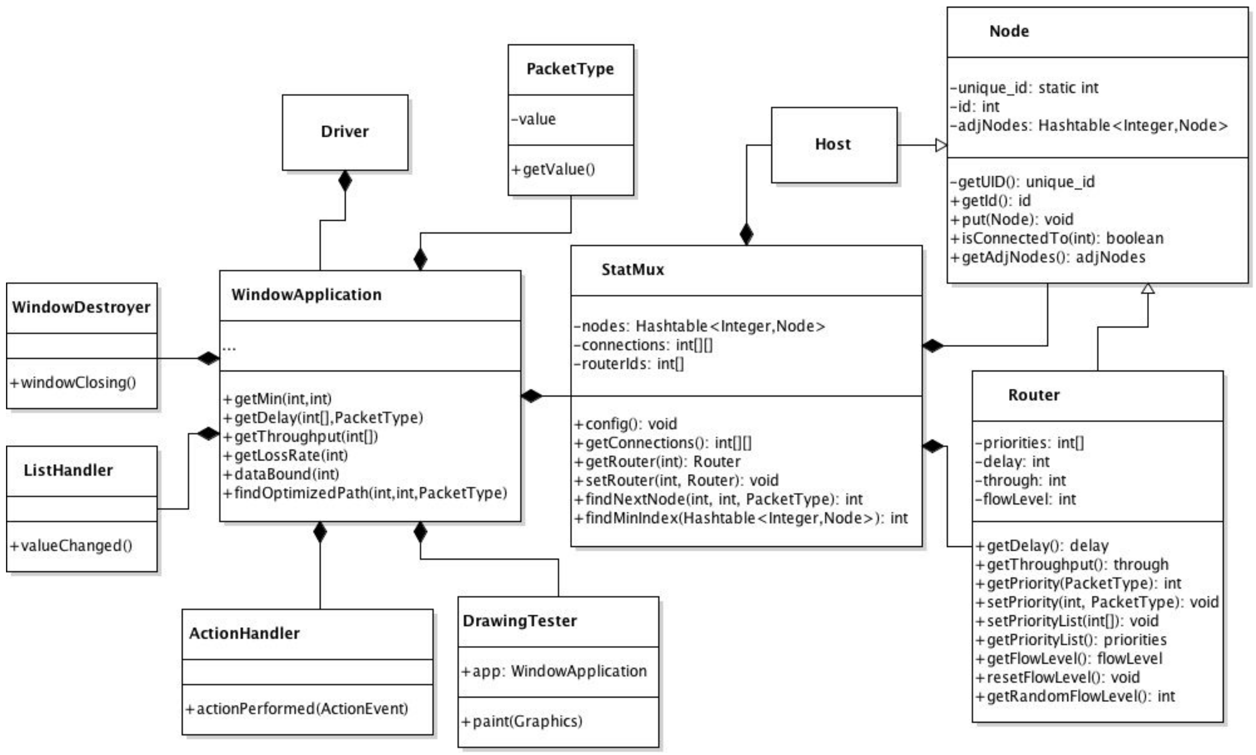


Fig. 3: class diagram of the project. It represents the classes and methods defined for simulating statistical multiplexer, and visualizing on a window application.

Class: Node
Description: represents a node in the network; it can be either a router or source/destination
Data field: ● unique_id: integer for holding node position in the network ● id: integer for holding node’s name ● adjNodes: a hash table for storing data of node’s neighbors
Methods: A. getUID ● type: int ● parameter: void ● return: unique_id of object B. getID ● type: int ● parameter: void ● return: id of object C. put ● type: void ● parameter: a Node object ● function: add an adjacent node into this node’s neighbor list. ● return: void D. getAdjNodes ● type: Hashtable object ● parameter: void ● function: return the adjacent neighbors’ list adjacent of the node ● return: hashtable of Node E. isConnectedTo ● type: boolean ● parameter: id of Node ● function: return true if parameter node is a neighbor of this node ● return: True if yes, else False

Class: Router
Description: represents a router in the network, extends class Node
Data field: ● priorities: array of integer for handling different packet flow types ● delay: integer represents delay at router ● through: integer represents through put at router ● flowLevel: integer represents congestion level at router
Methods: A. getDelay, getThroughput, getFlowLevel ● type: int ● parameter: void ● return: data field of Router object respectively B. getPriority ● type: int ● parameter: PacketType object ● return: priority for specific packet type C. setPriority, setPriorityList ● type: void ● parameter: priority type and value ● function: set priority based on parameter value ● return: void D. getRandomFlowLevel ● type: int ● parameter: void ● function: return a random number for congestion level at router ● return: int E. resetFlowLevel ● type: void ● parameter: none ● function: call getRandomFlowLevel for generating the flow level at router ● return: void

Class: Host

Description: represent a source/destination in the network, extends Node

Data field:

- id: integer represent host name

Methods:

A. getId

- type: int
- parameter: none
- return: id of Host object

Class: PacketType

Description: represents packet’s type value

Data field:

- value: priority value of packet type

Methods:

A. getValue

- type: int
- parameter: none
- return: enumerate value of PacketType object

Class: Packet

Description: represents a packet traversing the network

Data field (public):

- id: integer represent packet’s id
- src: integer represent packet’s source
- dst: integer represent packet’s destination
- size: integer represent packet’s size
- type: integer represent packet’s type

Class: StatMux
Description: simulates statistical multiplexer method of handling link sharing in network
Data field: ● nodes: Hashtable object holds Node objects in the network ● routersID: array of Router objects represents routers in the network ● connection: array of integers as buffer for backtracking connection between routers
Methods: A. config ● type: void ● parameter: void ● function: set up the network topology ● return: void B. getConnections ● type: int[][] ● parameter: void ● return: buffer for backtracking connection between routers C. getRouter ● type: Router ● parameter: id of router ● return: Router object with specified id D. setRouter ● type: void ● parameter: index, and Router object ● function: in Router array, replace Router object with specified index ● return: void E. findMinIndx ● type: int ● parameter: Hashtable object, and a key ● function: return index of router that will produce minimum delay for transferring packet, based on (key) steps ● return: index of the router

F. findNextNode

- type: void
- parameter:
    1. id1: current id of Node object
    2. id2: the destination of packet
    3. type: PacketType object represent the flow type
- function: Returns the next hop id, given the current and end ids with a packet type. The optimized next hop is chosen as one with the minimal effective delay, based on statistical multiplexer concept.
- return: index of next Node

Class: WindowApplication

Description: visualizes the statistical multiplexer concept on a window application

Data field:

- StatMux object: for simulating the statistical multiplexer concept
- PacketType object: represents packet type being traversing the network
- JLabel, JTextField, JList, JButton, JScrollPane objects for displaying purpose
- Loss, Delay: doubles for displaying data purpose
- Throughput: integer for displaying data purpose
- WindowDestroyer, ListHandler, ActionHandler, DrawingTester: private classes extended and implemented from library for displaying and interacting with the window application; the name implies respectively their purpose in WindowApplication class

Methods:

A. getMin

- type: int
- parameter: two integers
- function: return the value of lesser integer
- return: int

B. dataBound

- type: int
- parameter: int
- return: return integer value between [0,2] interval

C. getDelay

- type: int
- parameter: array of integers for packets' optimized traversing path, and packet type
- function: based on optimized path, backtracking the delay of packet traversing through the whole network.
- return: total delay packets took to travel from source to destination

D. getThroughput

- type: int
- parameter: array of integers for packets' optimized traversing path
- function: based on optimized path, backtracking the throughput of packet traversing through the whole network.
- return: throughput of network

E. getLossRate

- type: int
- parameter: integer for number of transfer
- function: backtracking and calculating packet lost traveling from source to destination
- return: total of packet loss occurred during the number of transfer

F. findOptimizedPath

- type: int[]
- parameter:
    1. src: id of source
    2. dst: id of destination
    3. type: PacketType object represent the flow type
- function: repeatedly looping to find next hop id by calling findNextNode of the StatMux class; upon receiving the next node id, the function will use that id as the new source. Upon completing the loop, the function will record the optimized path that packets of specific type traverse in the network from source to destination.
- return: array of integers with hop id values.

Private Classes:

A. WindowDestroyer:

- extends from WindowAdapter library
- windowClosing method: stop the program, if the window application is closed

B. ListHandler:

- implements ListSelectionListener
- valueChanged method: return list index selected by user

C. ActionHandler:

- implements ActionListener
- actionPerformed method: upon user pressing a button on the window application, applies changes on the display by calling repaint method of DrawingTester class

D. DrawingTester

- extends JComponent
- paint method: draw circles to display Node objects of the network, draw lines to display link between Node, draw red lines to display optimized path packet being traversing between source and destination.

IV. Results

A. Application's display setup:

1. The circle represents a Node in the network. The string on top displays the type and unique id of the Node respectively:
   - "S" represents source
   - "R" represents routers
   - "D" represents destination respectively.
2. In the Router Nodes, the number on top represents the delay, and the bottom represents congestion level at the router.
3. The black line represents connection between two Nodes. The red line represents the optimized path that packets will traverse through from source to destination.
4. User can specify a source and a destination for transferring packets by selecting from the list.
5. User can select the router from the scroll list, and change the priority base of each router for handling a specific flow type; 0 for highest and 2 for lowest priority.
6. User can change the flow type by selecting a packet type from the Packet Type list.
7. Loss, delay, and throughput value are displayed on the right side for each network setup.
8. For visual purpose, the routers are displayed in layers, but it is actually an arbitrary topology network; the link cost for each router is different.

B. Result screenshots and Explanation:

Simulation #1: Packet type Audio, Source 1, and Destination 1

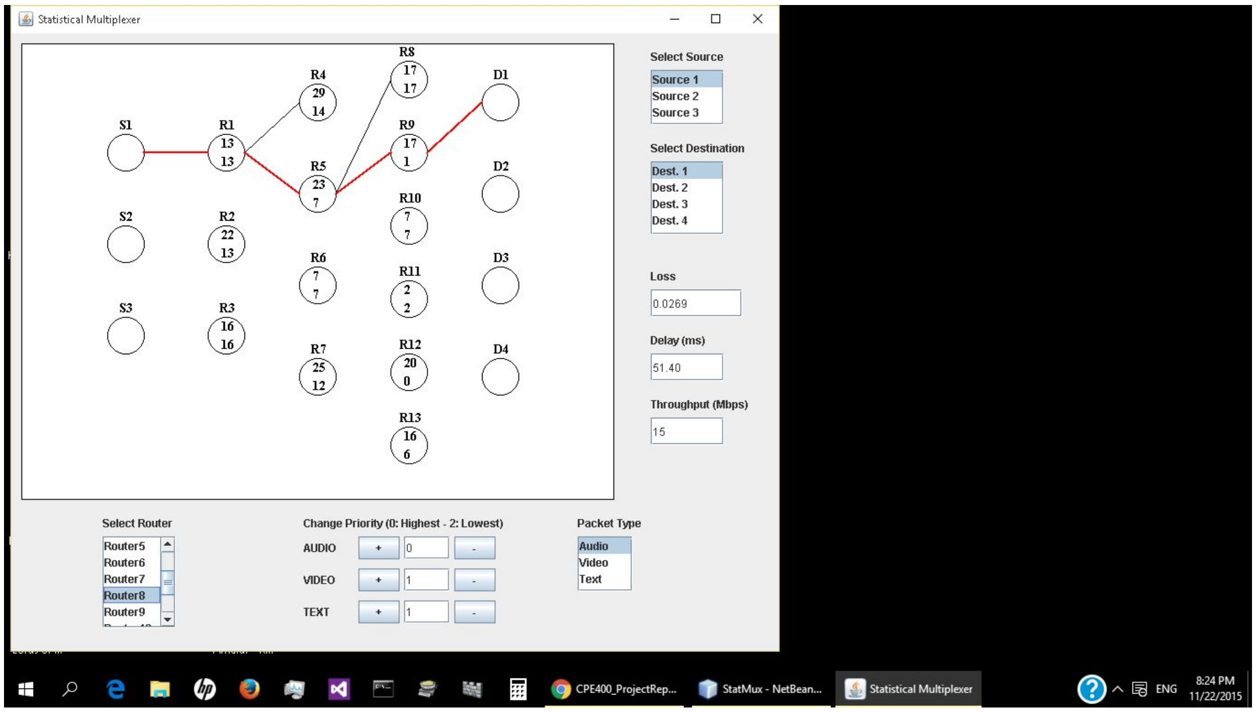
Source 1 only has connection with Router 1. Therefore, in the first hop, packets must travel from Source 1 to Router 1.

In the second hop, packets will travel to Router 5, because both the delay and congestion level are smaller than at Router 4.

In the next hop, the situation is trickier, because both Router 8 and Router 9 have same delay for transferring packet type Audio. However, the congestion level at Router 8 is much higher than that of Router 9; congestion of 17 in Router 8 compared to 1 in Router 8. The result is that packets will travel to Router 9.

The last hop, Router 9 sends packets to Destination 1, because there is none router left.

The behavior and display of the program are accurate, because statistical multiplexer also takes in account the congestion level.



Select Source: Source 1 (selected), Source 2, Source 3

Select Destination: Dest. 1 (selected), Dest. 2, Dest. 3, Dest. 4

Loss: 0.0269

Delay (ms): 51.40

Throughput (Mbps): 15

Change Priority (0: Highest - 2: Lowest)

AUDIO: 0

VIDEO: 1

TEXT: 1

Packet Type: Audio (selected), Video, Text

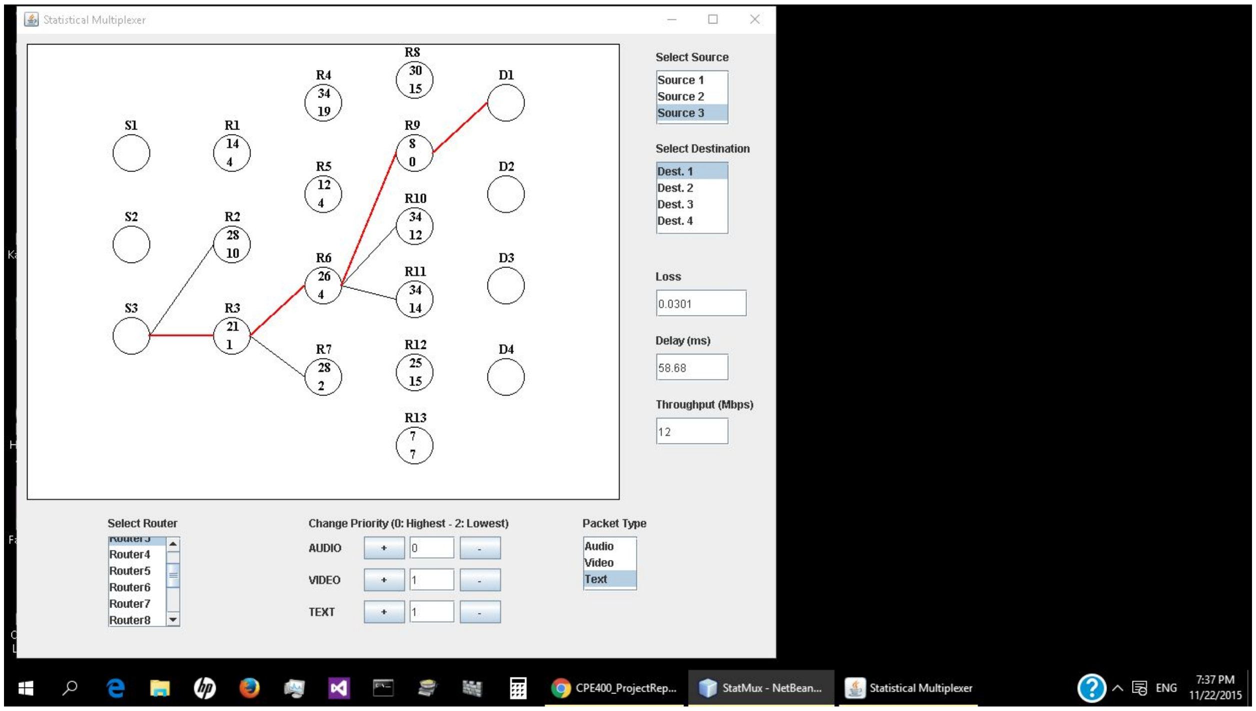
Simulation #2: Packet type is changed to Text, Source 3 selected, and Destination 1

Source 3 has connection with both Router 2, and Router 3. In the first hop, packets travel from Source 3 to Router 3, because both the delay and congestion level are smaller than at Router 2.

In the second hop, the situation is trickier, because Router 6 has slightly smaller delay, but it also has slightly higher congestion level compared to Router 7. The program was set to choose the path with smaller delay at router for solving these cases. Therefore, packets will travel to Router 6.

The next hop, packets will travel to Router 9, because both the delay and congestion level are smaller than the other two routers.

The last hop, Router 9 sends packets to Destination 1, because there is none router left.



Select Source: Source 1, Source 2, Source 3 (selected)

Select Destination: Dest. 1 (selected), Dest. 2, Dest. 3, Dest. 4

Loss: 0.0301

Delay (ms): 58.68

Throughput (Mbps): 12

Change Priority (0: Highest - 2: Lowest)

AUDIO: 0

VIDEO: 1

TEXT: 1

Packet Type: Audio, Video, Text (selected)

V. Conclusion and Future Works

Statistical multiplexer is a key concept in computer network. Statistical multiplexer finds an optimized path, for packets traversing between a source and destination, based on priority and need; the method also takes in account the difference in link cost, and network topology. Implemented the statistical multiplexer simulation project, we further understand the concepts, its advantage and disadvantage. For future works on this project, we will consider using Swing Timer in Java to handle the display, and we will also consider handling link failure for more realistic simulation.