University of Essex

9/14

Department of
Electronic Systems Engineering

Investigation of Quality of Service IP Routing Algorithm

Written By: Ivan Mwaipaja (Reg. No. 0211397)
M.Sc. Telecommunication and Information Systems

Supervisor: Dr. Sean Monaghan

Department of Electronic Systems Engineering
University of Essex
Colchester CO4 3SQ
Telephone: 01206 873333

University of Essex

Department of Electronic Systems Engineering

Student's full name: Ivan Nwaipaja

Scheme of Study MSc. Telecommunication and Information Systems

Title of Project Investigation of Quality of Service IP Routing Algorithm

FOR REFERENCE ONLY

Student's Certification

In accordance with Examination Regulation 6.6, I certify that I have acknowledged any assistance or use of the work of others in my project for the degree of MSc in the above scheme.

Signature: Ivan Nwaipaja

Date: 10th June 2003

This sheet, when completed, must be included as the first sheet in your project report.

Acknowledgements

The success of this project is the results of many people's efforts. In particular I want to express my appreciation to Dr. Sean Monaghan for his valuable, timely and enlightening comments and directives.

Many thanks should go to the University of Essex, department of Electronic Systems Engineering in particular for providing all the materials and equipment necessary for this project.

Last but not least, I am greatly indebted to my fellow students for being friendly and helpful throughout the project duration.

Ivan Mwaipaja.

Table of Content

Abstract

Internet communication is mainly determined by how best datagrams[1] are routed from source machine to destination machine. Routing is done in the network layer of the OSI/RM by special machines called routers. Using proper protocols[2] routers can compute a path over which a datagram has to follow to reach the desired destination.

One of the protocols most commonly used in routers is the OSPF. This protocol implements an algorithm called Dijkstra's algorithm[3] in computation of the relevant path to destination for an incoming datagram. Basically Dijkstra's algorithm uses only shortest paths to calculate routes; i.e. the algorithm will return shortest possible routes from a source router to all other routers in a network[4].

Dijkstra's algorithm does not consider the quality of service required by the individual datagrams; as long as they are destined to the same router they will all follow the same path. Hence, there is a need to have an algorithm that will take into account the QoS required by the individual packets. One of the algorithms that consider QoS is that described in [1].

In this project, the above mentioned algorithms will be simulated by writing Java computer program(s). Emphasis will be on investigating the QoS IP routing algorithm. The results of the two algorithms for different network conditions will be analysed and compared. A well GUI will be implemented to better illustrate the algorithms for the benefit of many users.

Finally, comments and conclusion will be drawn on how best the QoS IP routing algorithm can replace the existing Dijkstra's algorithm.

---

[1] In network layer the packets are called datagram due to the fact that routing is not reliable (best effort).
[2] Protocol can simply be defined as a set of instructions that enable devices to communicate. It's the language the devices speak.
[3] Throughout this report, Dijkstra's algorithm will be used to refer to the Dijkstra's shortest path algorithm used in OSPF.
[4] In this report a network refers to an interconnection of routers through which a shortest path is to be determined.

Chapter 1: Introduction

1.1: Organisation of the Report

This report is organised as follows.

In this first chapter the objective of the project are given together with general ideas on which the project is based on. Chapter 2 gives the theories of the java programming language and the algorithms that are to be used in this project. In chapter 3 a detailed process of developing and using the simulating program is presented and the results of using the program are given in chapter 4.

Finally, the implementation and results of the project are discussed and conclusions and recommendations provided in chapter 5.

1.2: Objective of the Project

The work reported in this project has been done with the following objectives.

i. To write a java program to find Dijkstra's paths for given networks. Necessary simplifications can be considered to make the Dijkstra's algorithm well implementable as a computer program. The program will input a specific network (routers, links and distances[5]). The program's output will be a kind of a routing table at the source router with shortest paths to the destination routers.

ii. To write a java program to find QoS routes for given networks and loading. Necessary simplifications can be considered to make the QoS IP routing algorithm well implementable as a computer program. The program will input a specific network (routers and links) and the bandwidth available on the links. The program's output will be a kind of a routing table at the source router with QoS paths to the destination routers.

iii. To Produce a GUI to demonstrate these programs. The GUIs are to be a user-friendly so that the programs can be easy to use and re-use.

iv. To compare the two programs. The programs will be applied to different networks with different conditions. The findings from the two programs will be analysed and compared to see how best the QoS IP routing algorithm will serve the new internet requirements.

[5] The term distance has been used for convenience purposes; it may not really refer to the geographical distance between two routers. Distance can refer to some other metrics like number of hops, mean queuing and transmission delay or even cost.

1.3: General Introduction

The Internet emerged from a network of some institutions (e.g. Universities, Military units) in the United States of America. Right now the use of internet has expanded to cover the whole world. It can easily be seen that not just the number of Internet users has increased but also the applications of the Internet have significantly increased. Earlier use of the Internet was just for simple document transfer while now the Internet can support audio and video transfer.

Since the internet was originally designed as a simple communication means, then real time/multimedia communication over the internet may experience some difficulties. They may not get the quality of service they require. While it is tolerable to have a delay of say 30 seconds for a document transfer, such a delay is completely out of way for real time/multimedia communication.

To solve this problem, the Internet needs to be re-engineered. One of the areas of interest for re-engineering is the protocols that are used to route packets from source router to destination router. The new protocol(s) should be able to fulfil the QoS requirements of the packets as opposed to the use of "best-effort" to route the packets to the desired destinations.

The current protocol(s) implemented in most of the routers uses distance to determine routes. This is to say for every datagram the protocols will find a route with shortest possible distance from the source and the datagram will follow this route. One of these protocols is the OSPF that implements Dijkstra's algorithm to compute the shortest paths to destination nodes[6].

The uses of Dijkstra's algorithm will give the shortest path but the path may not meet the required QoS. The most important determinant of QoS that will be focused on this project is the bandwidth required for efficient packet routing. The following are some of the scenarios to illustrate the failure of Dijkstra's algorithm to meet QoS requirements.

i. The Dijkstra's path may include links that do not have enough bandwidth to efficiently transfer the packets. This may result into packets being queued and cause delay or packets being discarded and cause data lost.

ii. Re-computation of Dijkstra's path may result into a path that can make packets to arrive at the destination with a different order as compared to the order with which the packets were sent. Although this can be corrected at the Transport layer, it causes unnecessary computations that in turn cause delays.

A new routing algorithm (QoS IP routing algorithm) introduced in [1] is to takes care of the failures of the Dijkstra's algorithm. The algorithm puts into considerations both distance and bandwidth availability on computing the routes to destination nodes. It will return routes with highest bandwidths for different distances (number of hops) from the destination nodes.

---

[6] Throughout this report routers in a network are also referred as nodes.

The algorithm is still experimental, so it needs to be tested before it can actually be implemented. There are several ways of testing such an algorithm. In this project the algorithm will be tested by writing a computer program (in Java language) to implement the algorithm and compare its functionality with another program that implement Dijkstra's algorithm.

1.4: Simplification Assumptions

In order to achieve the goal of producing Java programs to simulate Dijkstra's the bandwidth QoS algorithms, certain assumptions are considered to keep the program simple and still able to produce the required results.

Internet communication uses telecommunication links between one node and another. Each telecommunication link is associated with properties like capacity to transfer data (bandwidth) and metrics like distances and cost of using the link. Ideally, these properties and metrics are supposed to be the same in both directions of transmission (symmetric communication links). Practically (especially in wireless links) due to transmission system design or even just by agreed policies these properties and metrics are not always symmetric. In this project the links between nodes are modelled to be symmetric. Modification of the program to handle asymmetric links is possible with increase in program design and use.

When computing paths, the inter-network is represented by a graph of nodes and links. It should be noted that the nodes do not necessarily represent routers as it is normally the case. The nodes may also represent transit networks[7] and stub networks[8]. In cases where the graph has transit and stub networks special treatments like having zero-metric links have to be considered. In the interest of reducing program complexity the project assumes all nodes in the graph are routers. For the bandwidth QoS algorithm the extensions to include transit and stub networks are well explained in [1].

In the bandwidth QoS algorithm, effective use of link bandwidth and the legacy link metric have to be considered. In the algorithm used in this project (adopted from [1]) the legacy link metric that is considered is the simple hop count. The algorithm will find a path that meet QoS requirement with least hop counts. Other metrics could be considered but, as it can be clearly noted, with increase in program complexity which reflects the router's path computation complexity.

[7] Multi-access networks used to interconnect routers [1]
[8] Networks with only one connection to the network graph [3]

The algorithm is still experimental, so it needs to be tested before it can actually be implemented. There are several ways of testing such an algorithm. In this project the algorithm will be tested by writing a computer program (in Java language) to implement the algorithm and compare its functionality with another program that implement Dijkstra's algorithm.

1.4: Simplification Assumptions

In order to achieve the goal of producing Java programs to simulate Dijkstra's the bandwidth QoS algorithms, certain assumptions are considered to keep the program simple and still able to produce the required results.

Internet communication uses telecommunication links between one node and another. Each telecommunication link is associated with properties like capacity to transfer data (bandwidth) and metrics like distances and cost of using the link. Ideally, these properties and metrics are supposed to be the same in both directions of transmission (symmetric communication links). Practically (especially in wireless links) due to transmission system design or even just by agreed policies these properties and metrics are not always symmetric. In this project the links between nodes are modelled to be symmetric. Modification of the program to handle asymmetric links is possible with increase in program design and use.

When computing paths, the inter-network is represented by a graph of nodes and links. It should be noted that the nodes do not necessarily represent routers as it is normally the case. The nodes may also represent transit networks[7] and stub networks[8]. In cases where the graph has transit and stub networks special treatments like having zero-metric links have to be considered. In the interest of reducing program complexity the project assumes all nodes in the graph are routers. For the bandwidth QoS algorithm the extensions to include transit and stub networks are well explained in [1].

In the bandwidth QoS algorithm, effective use of link bandwidth and the legacy link metric have to be considered. In the algorithm used in this project (adopted from [1]) the legacy link metric that is considered is the simple hop count. The algorithm will find a path that meet QoS requirement with least hop counts. Other metrics could be considered but, as it can be clearly noted, with increase in program complexity which reflects the router's path computation complexity.

[7] Multi-access networks used to interconnect routers [1]
[8] Networks with only one connection to the network graph [3]

Chapter 2: Background

2.1: The Java Programming Language

(Based on an introductory chapter in [2])

Java is a relatively new but powerful programming language that is gaining popularity all around the world. The popularity is due to many reasons. One major reason is that it is principally *platform independent.* This means that a program written for one computer is very likely to run on another computer without any changes. This enables engineers to write programs that can work across all kinds of computers (PCs, Macs, UNIX workstations, etc.). This "write once, run anywhere" philosophy means an organisation that is to use java programs is not bounded to use a single type of computer systems.

The fact that Java is *object oriented* is another its advantage. Object oriented programming languages make the design and maintenance of large programs easier. Data and methods for modifying data are encapsulated into discrete units called **objects.** Object interact with each other through well defined interfaces this minimizes unintended side effects and make objects easily being re-used in different programs.

A third advantage of Java is that the basic language is relatively *simple.* The language has a simpler syntax than C (upon which it was based), making it easy to master. Java has omitted many of the trickiest and most error-prone portions of C language and simplified (or automatically handle) other features.

In addition, the standard java language includes *device-independent graphics.* Other programming languages like C and Fortran can create graphics but the code used is not standard, meaning that it may defer from computer to computer or even from application to application within the same computer. In contract a code written in java can generate the same graphics on two different computers even if they are of different type and/or use different operating systems.

Finally, Java is *free.* Java can be installed from the Java Development Kit (JDK) which is freely downloadable from the Internet (e.g. http://java.sun.com). The JDK includes a Java compiler (javac), a Java run-time interpreter (java), a debugger (jdb), and all of the standard libraries. Fancier development environments can be purchased from different vendors but the basic language is free.

Java has one common disadvantage. Since it is an evolving language, the Java Application Programming Interface (API) is rapidly changing. This makes features of programs that were written in older versions of JDK become deprecated (though the basic language is fixed). Hopefully, the core portion of the java API will soon mature and stabilize so programmer can work with a consistent environment. In this project the programs written uses Java API as it appears in Microsoft Visual J++ 6.0.

Elements of Java

Java is made up of three distinctive elements:

1. The Java Programming Language
2. The Java Virtual Machine
3. The Java Application Programming Interface (API)

Java is different from other languages in that all java programs are compiled t execute on a special computer environment called *Java Virtual Machine* (JVM). The machine language of JVM is called *bytecodes*. When a java program is compiled it produces bytecodes, which can be executed directly on a Java Virtual Machine as opposed to processors of real computers. Each type of computer has an interpreter that converts the bytecodes into machine language of a particular computer as a Java program is executed.

A Java program may be created using any text editor, and saved in a file with an extension of *.java*. When this program is compiled, the JVM will store the bytecodes in a file with same name but the extension will now be *.class*. During the execution of the java program, the Java interpreter translates the bytecodes into the actual instructions to be executed. Note that the Java bytecodes is independent of any particular computer hardware, so any computer with a Java interpreter can execute the compiled Java program, no matter what type of computer the program was compiled on. Figure 2.1 below illustrate the process.

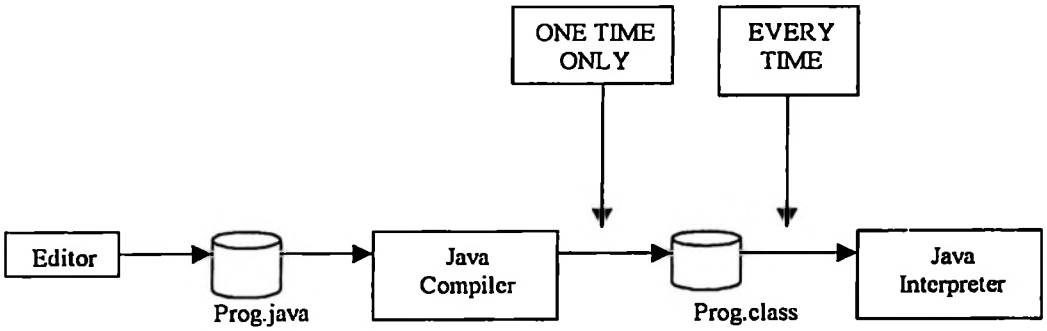


Figure 2.1. Creation, Compilation and Interpretation of Java Programs.

The Java API is a large collection of ready-made software components that provide many useful capabilities. These components are grouped into libraries of related components called *packages* and can provide many functionalities including manipulation of strings, build GUI, and many more. Enormous amount of time can be saved by using these standard packages in programming rather than trying to "reinvent the wheel" each time a program is written. The components in the packages are standard across all implementations of Java and they are assured of being bug-free. Hence, they can be used in any computer that implements Java with no need of debugging.

In this project, emphasis will be on using these Java API standard components so as to make the programming easy and less error-prone.

2.2: Dijkstra's Shortest Path Algorithm

The main function of the network layer (of OSI model) is to route packets from source machine to destination machine [3]. This is done by special computers called routers that implement routing algorithms to decide which output line an incoming packet should be transmitted.

When the decision above is based on the shortest distance between the source machine and the destination machine, Dijkstra's algorithm can best serve the purpose. The algorithm will return the shortest distances from the source router and all the existing routers on a network. An advanced use of the algorithm can also return the actual paths across the network that span the shortest distances described earlier.

The term shortest distance has been used for convenience purposes; it may not really refer to the geographical distance between two routers. Distance can refer to some other metrics like number of hops, mean queuing and transmission delay. Cost metric can also be used when link[9] use are charged, in which case the shortest distance path will mean a path that will consume least amount of money.

The algorithm starts by considering a graph (network) with each node on the graph representing a router and each arc of the graph representing a link of certain length (or any other metric value). For a given router, Dijkstra's algorithm can be used to find shortest distances from the router and the other routers on the network and indeed the actual paths. The algorithm works as follows [4]:

1. Separate the nodes into two sets: the set of evaluated nodes, E, for which the shortest paths are known, and the set of remaining nodes, R. Also maintain an ordered list of paths, O.
2. Initialise the set E to contain only the source node S and R to contain all the other nodes. Initialise the list of paths O to contain the one segment paths starting from S. Each of these paths has a cost equal to the corresponding link's metric. Sort list O by increasing metrics.
3. If the list O is empty, or if the first path in O has an infinity metric, mark all nodes left in R as unreachable. The algorithm has terminated.
4. First examine P, the shortest path in list O. Remove P from O. Let V be the last node in P. If V is already in set E, continue at step 3. Otherwise, P is the shortest path to V. Move V from R to E.
5. Build a set of new candidate paths by concatenating P and each of the links starting from V. The cost of these paths is the sum of the cost of P and the metric of the link appended to P. Insert the new links in the ordered list O, each at the rank corresponding to its cost. Continue at step 3.

If the Dijkstra's algorithm is adequately implemented it converges in order of MxlogM iterations while another popular algorithm, Bellman-Ford algorithm converges at the order of NxM where M and N are the number of links and nodes respectively. For a large network, it can clearly be seen that Dijkstra's algorithm is far more efficient and hence it has been recommended to be used in OSPF[10] protocol.

---

[9] This refers to a communication channel between routers

[10] Open Shortest Path First, a protocol used for interior gateway routing (within an autonomous system)

Example on how Dijkstra's Algorithm works
(Based on a question in [5])

Consider the network shown in figure 2.2 below with A as a source node and link costs as shown.

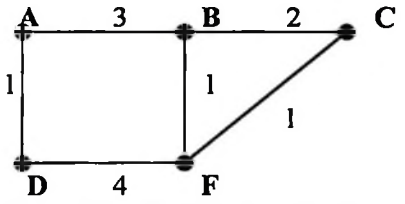


Figure 2.2: Network of routers to demonstrate how Dijkstra's Algorithm works.

- Initial State(step 2 above):
  E = {(A, AA, 0)} **R** = {B, C, D, F} **O** = {(AD, 1), (AB, 3)}

- State 2: Examine path AD. D is not in **E**. Move D from **R** to **E**.
  E = {(A, AA, 0), (D, AD, 1)} **R** = {B, C, F}
  O = {(AB, 3), (ADF, 5)}

- State 3: Examine path AB. B is not in **E**. Move B from **R** to **E**.
  E = {(A, AA, 0), (D, AD, 1), (B, AB, 3)} **R** = {C, F}
  O = {(ABF, 4), (ABC, 5), (ADF, 5)}

- State 4: Examine path ABF. F is not in **E**. Move F from **R** to **E**.
  E = {(A, AA, 0), (D, AD, 1), (B, AB, 3), (F, ABF, 4)} **R** = {C}
  O = {(ABC, 5), (ADF, 5), (ABFC, 5), (ABFD, 8)}

- State 5: Examine path ABC. C is not in **E**. Move C from **R** to **E**.
  E = {(A, AA, 0), (D, AD, 1), (B, AB, 3), (F, ABF, 4), (C, ABC, 5)}
  R = { } **O** = {(ADF, 5), (ABFC, 5), (ABCF, 6), (ABFD, 8)}

- State 6 to 9: Since all nodes are in **E** and **R** is empty, the paths in **O** will be removed from **O** and not added to **E**. So, at the end of step 9 E and will remain unchanged and **O** will be empty.

- Final State: Based on information from E, the source node, A may generate a routing table that looks like below.

Destination	Path	Distance
A	AA	0
B	AB	3
C	ABC	5
D	AD	1
F	ABF	4

2.3: Introduction to Internet Quality of Service

Quality of service is a new terminology in Internet use which may be difficulty to interpret so one has to make it clear before he/she further uses the term. It can be defined as a set of service requirements to be met by the network while transporting a flow[11] [6]. It is a measurement of the network behaviour with respect to certain characteristics of defined services[12].

According to the definition in [6] above, the service requirements can also be called the determinants of QoS. In Internet communication we have four basic determinants of QoS namely, *reliability*, *delay*, *jitter*, and *bandwidth*. Below are explanations of the determinants based on a table in figure 2.3 which gives some common applications and the stringency of their requirements.

Reliability in a network is a measure of how much a network is capable of handling a flow without introducing any errors. The first four applications require stringent reliability as accurate delivery of data is very important. On the other hand, audio or video applications may work with low reliability.

Delay is the measure of time taken for a network to transmit a packet from a source to a destination. Real-time applications, such as telephony and videoconferencing have strict delay requirements. If a network will introduce an exact delay of say 2 seconds the telephony users will not tolerate this but audio/video-on-demand users can, since the applications are not delay sensitive and same it is for e-mail and file transfer. Interactive applications, such as Web surfing and remote login, are slightly more delay sensitive.

Jitter is variation in packet arrival times. Video and audio are extremely sensitive to jitter. If transmission time varies, the results at the receiver will be terrible. The first three applications are not sensitive to jitter. Remote login is somewhat sensitive to that, since characters on screen will appear in bursts.

Bandwidth is the measure of amount of data that can be transmitted in a given time. Different applications have different bandwidth requirement, with e-mail and remote login not needing much, but video in all forms needing a great deal.

Application	Reliability	Delay	Jitter	Bandwidth
E-mail	High	Low	Low	Low
File transfer	High	Low	Low	Medium
Web access	High	Medium	Low	Medium
Remote login	High	Medium	Medium	Low
Audio-on-demand	Low	Low	High	Medium
Video-on-demand	Low	Low	High	High
Telephony	Low	High	High	Low
Videoconferencing	Low	High	High	High

Figure 2.3. How stringent the QoS requirements are [3]

[11] In Internet use flow refers to a stream of packets from a source to a destination.

[12] A service means something offered to the final user, e.g. end-to-end communication, client server application, data transport, etc.

Out of the four requirement explained above, bandwidth seem to be the dominating one. One of the reasons for its dominance is that it can be implemented in routing algorithms. One of the routing algorithms that consider bandwidth for QoS assurance is the one stated in [1]. In this project, only bandwidth will be considered for QoS requirement.

2.4: Bandwidth QoS Routing Algorithm
(Based on the algorithm in [1])

With the existing routing protocols (e.g. OSPF) implementing shortest path based (least cost) algorithms not able to fulfil the QoS required for today's Internet communication, new algorithms have to be introduced to solve the problem. This small report describes a path selection algorithm, which for a given network topology and link available bandwidth, pre-computes all possible QoS paths[13], while maintaining a reasonably low computational complexity.

Specifically, the algorithm pre-computes for any destination a minimum hop count path with maximum bandwidth, i.e., the path cost is a function of its available bandwidth. Available bandwidth of a path means the smallest available bandwidth on all links of the path (bottleneck link bandwidth). At the k^{th} iteration (equals to k hop counts) of the algorithm, the maximum bandwidth available to all destinations on a path of no more than k hops is recorded.

Description of the Algorithm

1. The algorithm pre-compute paths from source node to all possible destination nodes and for all possible bandwidth values. At each iteration (hop count), intermediate results are recorded in a QoS routing table.

2. The QoS routing table is a KxH matrix, where K is the number of destinations (nodes in the graph) and H is maximum allowed number of hops[14].

3. The (n;h) entry of the routing table is built during the h^{th} iteration of the algorithm. It has two fields. The first is *bw* which is the maximum available bandwidth, on a path of at most h hops between the source node and destination node n. The second is *path*, the list of all the nodes traversed by the path from source node to destination node.

4. When the algorithm is starts, the routing table is first initialised with all *bw* fields set to 0 and the *path* fields cleared.

5. The entries for the one-hop paths are modified in the following way: The *bw* field is set to the value of the bandwidth available on the direct link from the source. The *path* field is set to source node and the identity of the next node which in this case is the destination.

---

[13] A small modification has been done from the original algorithm in [1] that computes only the next hop (neighbour).
[14] Can be network diameter (implicit) or less (explicit) for worst case complexity control.

6. At other iteration h, the algorithm looks at each link (n; m) where n is a node whose bw value has changed in the previous iteration and record the minimum between the bw field in entry (n; h-1) and the link metric value b(n; m), the available bandwidth on the link from node n to m.

7. If this value is higher than the present value of the *bw* field in entry (m; h), it is recorded together with the this path from the source node to m.

After the algorithm terminates, the routing table provides for all destinations and bandwidth requirements, the path with the smallest possible number of hops and sufficient bandwidth to accommodate the new request. Furthermore, this path is also the one with the maximum available bandwidth among all the feasible paths with at most these many hops.

Upon arrival of QoS packet the routing table is used to find the QoS path (shortest path with highest bandwidth) as follows. The router looks at the row that corresponds to the destination node from leftmost entry (h = 1) to the rightmost entry (h = H). The first entry that fulfils the bandwidth QoS requirements will be considered for path selection. If none of the entries fulfils the requirement then the packet can not be routed with the required bandwidth QoS.

Example on how the Algorithm works
(Based on a question in [8])

Consider the network shown in figure 1 below with R as a source node and link available bandwidths as shown.

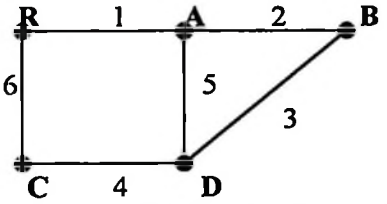


Figure 2.4: Network of routers to demonstrate how the QoS-based routing algorithm works.

Below is the table to show how the routing table of the source node, R, can be computed from the algorithm above.

Hops / Nodes	h = 1	h = 2		h = 3		h = 4	
	Actual		Actual		Actual		Actual
A	(1,RA)*		(1,RA)	(4,RCDA)	(4,RCDA)*	(2,RCDBA)	(4,RCDA)
B	0	(1,RAB)	(1,RAB)*	(3,RCDB)	(3,RCDB)*	(2,RCDAB)	(3,RCDB)
C	(6,RC)*		(6,RC)		(6,RC)		(6,RC)
D	0	(1,RAD) (4,RCD)	(4,RCD)*	(1,RABD)	(4,RCD)		(4,RCD)

Figure 2.5: Computation of QoS-based paths. (* means the entry has changed)

- For each iteration h, the first column shows the new paths to be considered. The second column shows the evaluated path.

- h = 0: The algorithm is invoked. Corresponds to step 4 (Not shown in the table).

- h = 1: The entries for the one-hop paths are modified. Corresponds to step 5.

- h = 2: Second iteration. We look at the nodes whose bw values has changed in the first iteration (A and C) and carry out as in step 6 and 7.

- h = 3: Third iteration. We look at the nodes whose bw values has changed in the second iteration (B and D) and carry out as in step 6 and 7.

- h = 4: Fourth iteration. We look at the nodes whose bw values has changed in the third iteration (A and C) and carry out as in step 6 and 7.

Chapter 3: Implementation

3.1: Dijkstra's Algorithm Program

Program Inputs

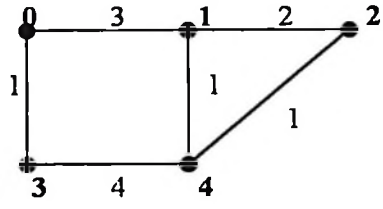
In order to generate the shortest paths to the destinations, full information on the network topology and the metrics associated with the links is an important input. This means that the program will require as its input all the nodes (by names) in a network, the way they are connected (possible links) and the lengths of the links.

To have a command-driven program with such inputs can be tedious to both the programmer and the user of the program. The following was done to simplify matters.

Firstly, all nodes in a network will be named by integer numbers starting from zero. This simplifies the exercise of input all the possible nodes to just the *number of nodes in a network*. For example if the input for the number of nodes in a network is 5, then the nodes should be named 0, 1, 2, 3, and 4. It's up to the program user to determine which of these numbers represent which node (by their actual names). Slight modification on the program can help this but that will mean more inputs to the program depending on the network size.

Secondly, the program assumes each node is connected to all the other nodes; that is for any two nodes in a network there is a link. For this, the program will require the *lengths (or costs) of each of the links* as a positive real number. For nodes that are not connected (no link between them) the input will be an infinity length (or cost). In this particular program infinity input is represented by "I" or "i".

For a network with N nodes the program will require N(N-1)/2 such inputs. Figure 1 below shows a network of 5 nodes (a) and its corresponding program input (b).



From / To	0	1	2	3	4
0	##	~~	~~	~~	~~
1	3	##	~~	~~	~~
2	I	2	##	~~	~~
3	1	I	I	##	~~
4	I	1	1	4	##

Figure 3.1 (a) The network (b) The inputs

Key:

##: means the input is not required as a link cost from a node to itself is obviously zero

~~: means the input is not required for a symmetric links[15].

[15] Only symmetric links have been considered in this program, these are links with same cost for both directions.

Since the algorithm works at a single node then another necessary input is the *source node* represented as an integer value (0 to 4 for a 5 node network). The source node is the node where the algorithm will be carried out. After the network topology is input then some source nodes can be chosen one at a time and corresponding outputs (routing tables) obtained.

Program's Flow Chart

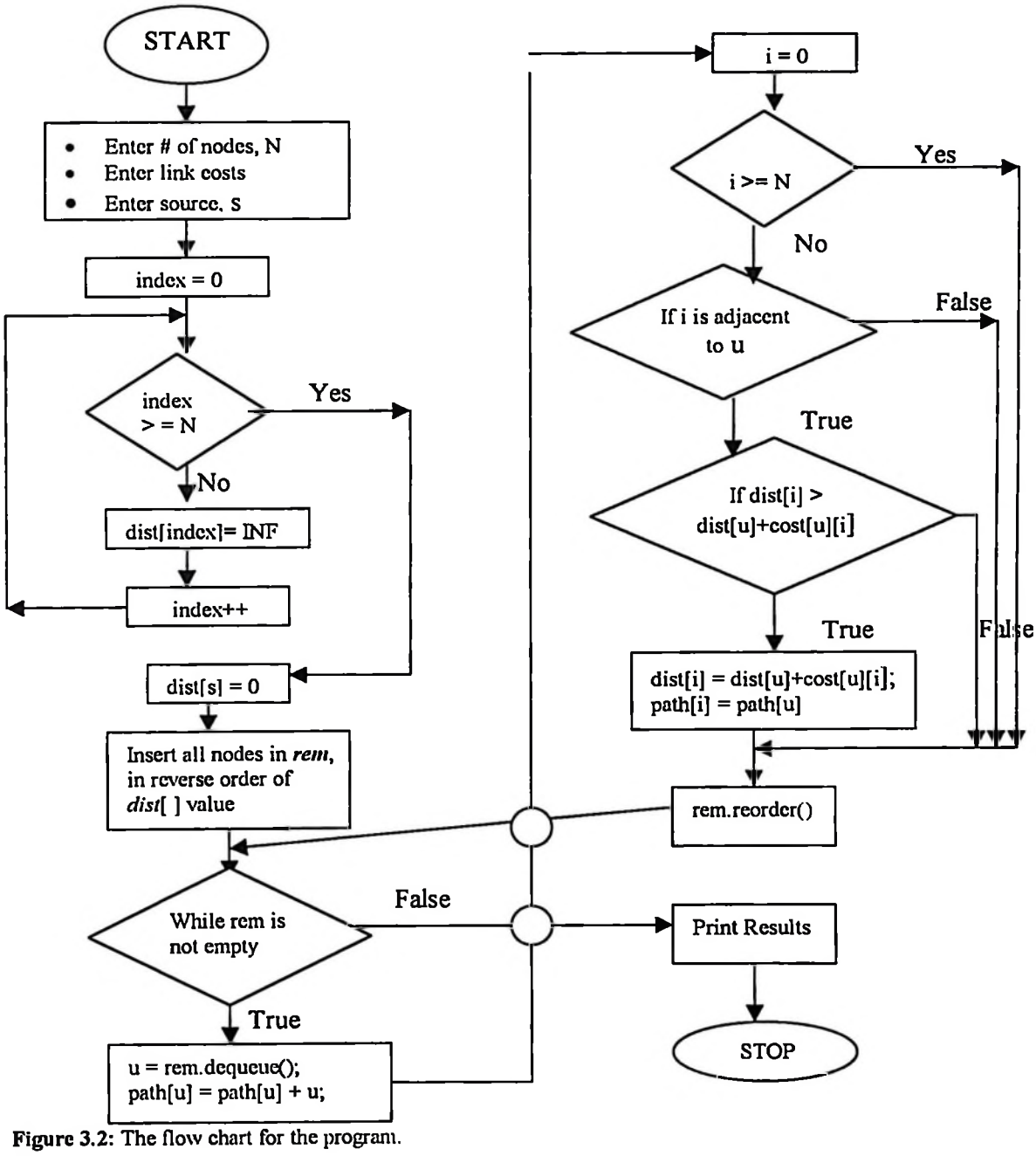


Figure 3.2: The flow chart for the program.

The flow chart above gives the logical structure of the program based on notations proposed in [7]. The following are explanations to some of the words used in the chart.

- ***dist***: array holding shortest distances from source s.
- ***rem***: priority queue of unvisited nodes prioritised by shortest recorded distance from the source.
- ***rem.reorder():*** reorders ***rem*** if the values in ***dist*** change.
- ***rem.dequeue():*** remove the first node from the ***rem.***
- ***path:*** array holding shortest paths from source s. Initially it is empty.
- ***cost[a][b]:*** array holding costs of all links a to b.
- ***Results:*** routing table consisting of the nodes and corresponding values from ***dist*** and ***path.***

Pseudocode:

```
For each node v {
        dist[v] = INFINTY;
        path[v] = "";
        dist[source] = 0;
}
Queue rem = new Queue();
Insert all nodes in rem in reverse order of dist[] values;

while (!rem.isEmpty()) {
        u = rem.dequeue();
        path[u] =path[u] +"u";
        for each v in rem adjacent to u{
                if(dist[v]>(dist[u] + cost[u][v])) {
                        dist[v] = dist[u] + cost[u][v];
                }
        }
rem.reorder();
}
```

How the Program Works

When the program is started a user is prompted to enter all the inputs relevant to the network topology. This includes the number of nodes and the cost of each of the links in a network. Also the user has to enter the source node. Figure 3.3 and 3.4 below show inputs and outputs (respectively) for the network in figure 3.1.

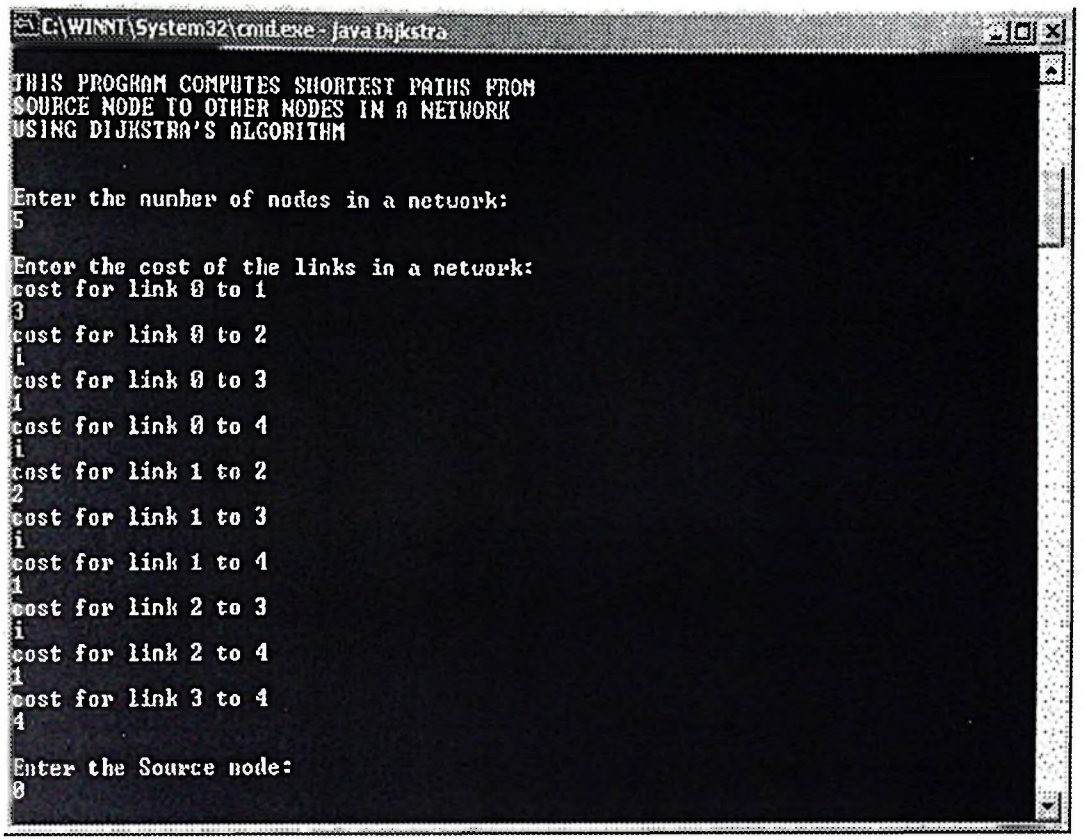


Figure 3.3: Program inputs.

After the input the algorithm will be carried out and the result delivered. It is important for a source node to have both the shortest distance to other nodes and the actual paths. This program will produce a routing table with destination nodes, shortest distances to the nodes and shortest paths to the nodes.

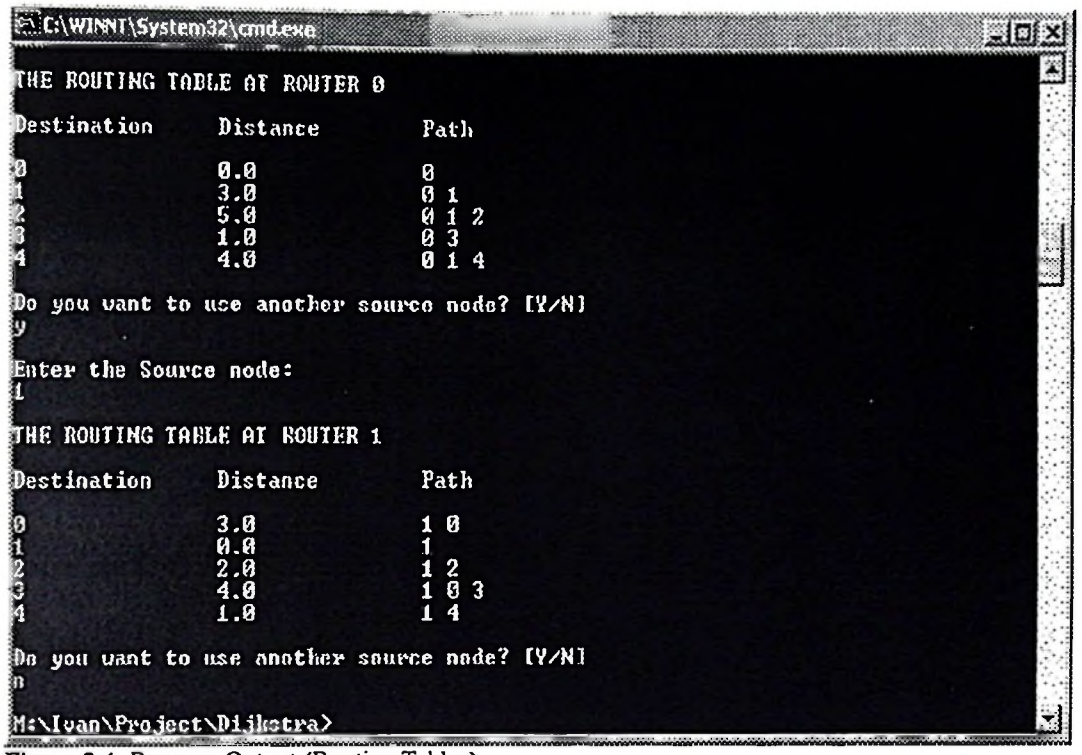


Figure 3.4: Program Output (Routing Tables)

3.2: Bandwidth QoS Routing Algorithm

Program Inputs

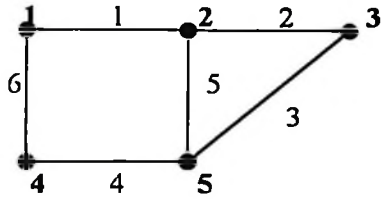
Just like for shortest path computation, full information on the network topology and metrics associated with the links are the basic input for the program to calculate QoS based paths. In this case the program will require all the nodes in a network, the possible links (nodes connection) and the available bandwidths of the links. To obtain these requirements the user will have to input the following.

Firstly, the number of nodes in a network has to be known to the program. This number will be used to estimate the memory (arrays, queues, and lists) required to handle the network information. This will also serve to get the names of all the nodes as the nodes will be named by integer numbers starting from 1 for simplicity. For example if the number of nodes in a network is 5, then the nodes will be named 1, 2, 3, 4 and 5. It's up to the program user to determine which of these numbers represent which node (by their actual names). Slight modification on the program can help this but that will mean tedious inputs to the program if the network is large.

The program assumes each node is connected to all the other nodes; this means for any two nodes in a network there is a link. Therefore, the program will require the available bandwidths of each of the links as a positive real number. For nodes that are not connected (no link between them) the input shall be zero bandwidth.

For a network with N nodes the program will require N(N-1)/2 such inputs. Figure 3 below shows a network of 5 nodes (a) and its corresponding program input (b).

Figure 3.5



From / To	1	2	3	4	5
1	##	~~	~~	~~	~~
2	1	##	~~	~~	~~
3	0	2	##	~~	~~
4	6	0	0	##	~~
5	0	5	3	4	##

(a) The network (b) The inputs

Key:
##: means the input is not required as a link available bandwidth from a node to itself is obviously infinity.
~~: means the input is not required for a symmetric links[16].

Another necessary input is the source node, the node where the algorithm will be carried out. The source node is represented as an integer value as explained above.

Lastly, the program requires the maximum number of hops that will be considered for path computation. It can be seen that for N number of nodes, the network diameter is N-1, which corresponds to the path that includes all the nodes. The program allows a user to explicitly specify any number less or equal to this. This makes the program flexible since the user may like to limit the path length to reduce computational complexity or reflect the network topology.

Pseudocode:

Note: The pseudocode below assumes an explicit route approach in specifying all intermediate nodes to the destination. The pseudocode also does not handle equal bandwidth multi-paths for simplicity, but the modification needed to add this support are straightforward.

[16] Only symmetric links have been considered in this program, these are links with same bandwidth for both directions.

Input:

num_of_vert = number of vertices (nodes) in a graph
vertex[] = array of vertices, labeled by integer from 1 to num_Of_vert
source = source vertex at which the algorithm is done
bwidth[] = array of available bandwidths on all possible links;
bwidth[n,m] means available bandwidth on a link between n and m.
maxHops = maximum hop-count (at most the network diameter)

Type:

tab_entry: record (Class in Java)
bw = integer,
neighbor = integer 1..num_of_vert,
path = String.

Variables:

TT[1..N, 1..maxHops]: topology table, whose (n,h) entry is a tab_entry record, such that:
TT[n,h].bw is the maximum available bandwidth (as known thus far) on a path of at most h hops between source node and n,
TT[n,h].neighbor is the first hop on that path (a neighbor of source node).
TT[n,h].path is a maximum-bandwidth-path of at most h hops between source node and n,

S_prev: list of vertices that changed a bw value in the TT table in the previous iteration.
S_new: list of vertices that changed a bw value (in the TT table etc.) in the current iteration.

The Algorithm:

```
begin;
 for n:=1 to num_of_vert do  /* initialization */
 begin;
  TT[n,0].bw := 0;   TT[n,0].neighbor := null;    TT[n,0].path := source;
  TT[n,1].bw := 0;    TT[n,1].neighbor := null;    TT[n,0].path := source;
 end;
 TT[source,0].bw := infinity;
 reset S_prev;
 for all neighbors n of source do
 begin;
  TT[n,1].bw := max( TT[n,1].bw, bwidth[source,n]);
  if (TT[n,1].bw = bwidth[source,n]) then
  begin
   TT[n,1].neighbor := n;
   TT[n,1].path := TT[n,1].path concatenate "n";
  end;
       /* need to make sure we are picking the maximum */
       /* bandwidth path for routers that can be reached */
```

```
 S_prev :=  S_prev union {n}
        /* only a router is added to S_prev, */
        /* if it is not already included in S_prev */
 end;


 for h:=2 to maxHops do   /* consider all possible number of hops */
 begin;
  reset S_new;
  for all vertices m in Vertex[] do
  begin;
    TT[m,h].bw := TT[m,h-1].bw;
    TT[m,h].neighbor := TT[m,h-1].neighbor;
    TT[m,h].path := TT[m,h-1].path
  end;

  for all vertices n in S_prev do
  begin;
    for all edges (n,m) do
    if min( TT[n,h-1].bw, bwidth[n,m]) > TT[m,h].bw then
    begin;
      TT[m,h].bw := min( TT[n,h-1].bw, bwidth[n,m]);
      TT[m,h].neighbor := TT[n,h-1].neighbor;
      TT[m,h].path := TT[n,h-1].path concatenate "m";
      S_new :=  S_new union {m} /* m has changed bw value */
    end
  end;
  S_prev := S_new;
        /* new iteration, s_new becomes s_prev */
  if S_prev=null then h=maxHops+1   /* if no changes then exit */
 end;

end.
```

How to use the Program

When the program is started a user is prompted to enter all the inputs relevant to the network topology. This includes the number of nodes, the available bandwidths of each of the links in a network and the source node. Also the user has to explicitly enter the maximum number of hops. Figure 4 below show inputs for the network in figure 3.

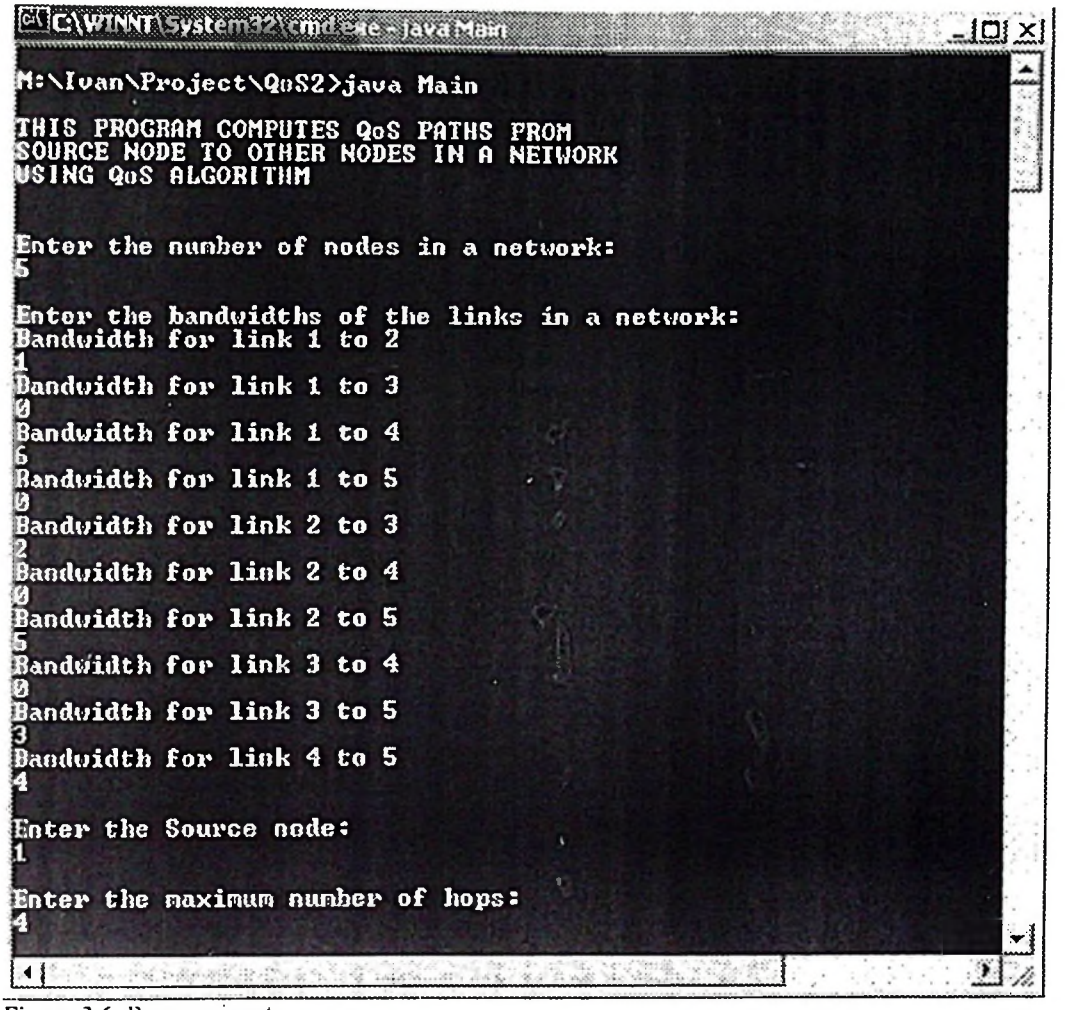
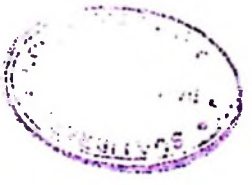


Figure 3.6 Program inputs

After the input, the algorithm will be carried out and the result delivered. This program will produce a routing table with destination nodes, maximum path bandwidths to the nodes and bandwidth QoS paths to the nodes. Figure 5 below shows the output of the program corresponding to the network in figure 3.



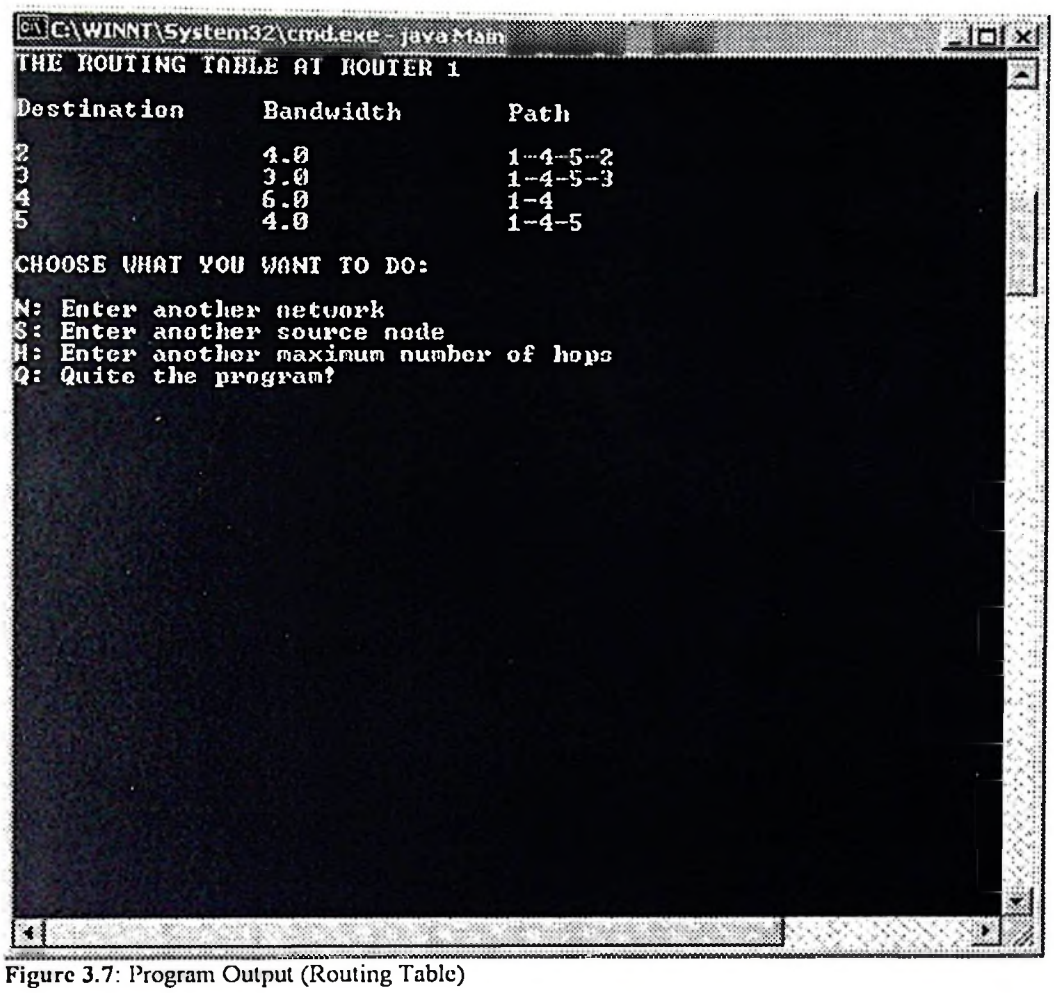


Figure 3.7: Program Output (Routing Table)

The program allows the user to change the source node and/or maximum number of hopes and obtain the corresponding results without having to re-input the network topology and bandwidth information. Also the program is repetitive, that is a new network may be input without having to quit and restart the program.

3.3: Graphics Improvements

(The Graphics Design was adopted from [11])

After having the programs (Dijkstra's and Bandwidth QoS algorithms) running in a command driven interface it was better to improve these programs to have a Graphical User Interface (GUI). The idea is to present a pictorial interface to a program and give a program a distinctive "look" and "feel" [9]. It has been able to do this since Java has built-in graphics features that are platform-independent.

Though Java is rich in graphics features, but as far as this project is concerned just some of them were used. The table below summarizes the features used and their descriptions.

Feature	Description
JLabel	An area where uneditable text or icon can be displayed
JTextField	An area in which the user inputs the data from the keyboard. The area can also display information.
JButton	An area that triggers an event when clicked.
JComboBox	A drop-down list of items from which user can make a selection by clicking an item in the list or by typing into the box, if permitted.
JPanel	A container in which components can be placed.

Figure 3.8: Some GUI features used in this project [9]

Finally, the GUI was implemented as an applet[17] to make it easy to access. With an applet the program can be stored on the web server and easily be accessed world wide through the internet by using web browsers like Internet Explorer, Netscape Communicator, etc.

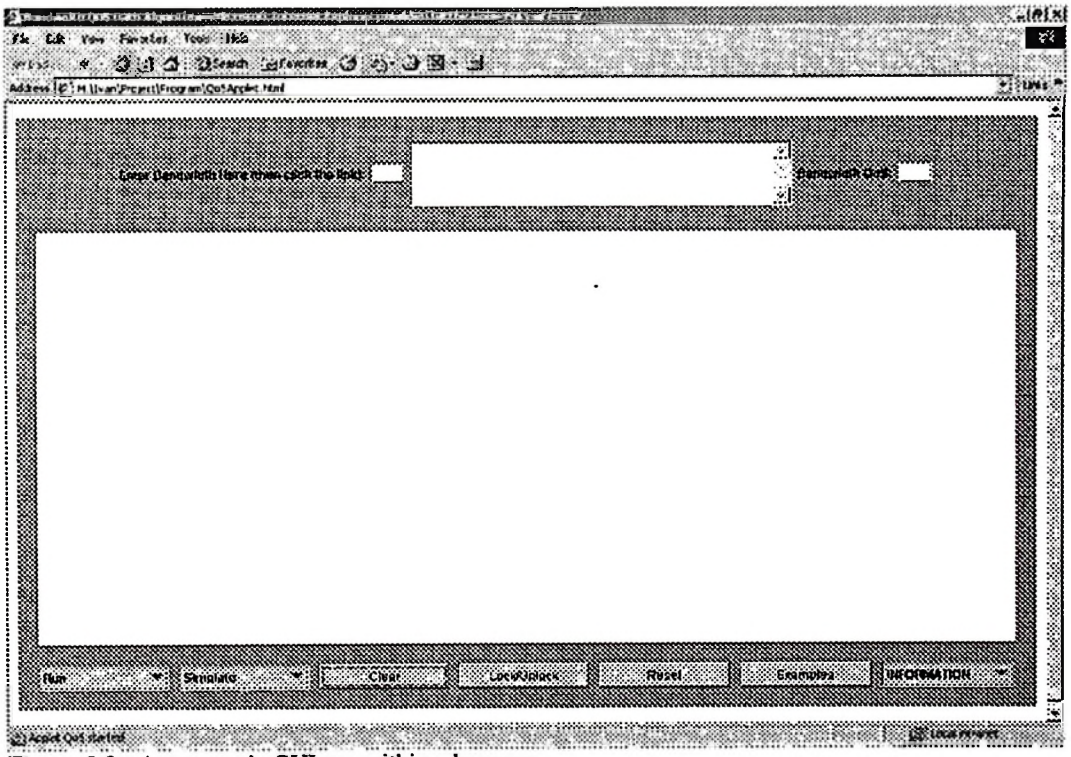


Figure 3.9: A program's GUI run within a browser.

How to use the Program

As stated earlier the programs were implemented as applets, so they can be accessed by using web browsers that are available in almost every computer. All a user has to do is to start the browser and type the URL to where the program files are. For example: *A:\project\QoSApplet.html.*

[17] An applet is a special type of Java program that is designed to work within a WWW browser [2]

After the program is started, the network to be simulated is to be drawn on the empty space provided. A network will consist of nodes, links and the associated bandwidth values. The nodes are drawn simply by clicking where on the space provided the node is intended to be with the first node to be drawn as source node.

The links between nodes can be drawn by dragging the mouse form one node to another. Once the link is drawn it is assigned a default bandwidth value that can be changed to any integer value. The process of changing the bandwidth values is done by typing the new bandwidth values in the space provided and clicking on the centre of the relevant links.

When a network is completely drawn, it is ready to simulate different flows originating from the source node to various chosen destination nodes. For each flow, destination node and the required bandwidth value have to be specified. The required bandwidth value is specified by typing on the space provided and the destination node is specified by right clicking the required node.

On running the algorithm, it will find and establish the best paths that meet the bandwidth QoS required to the destinations. This is so even for the program that uses Dijktsra's algorithm which will produce the same path to a destination node for every bandwidth QoS specified. If this path doesn't meet the bandwidth QoS the destination is blocked.

The programs keep track of the available link bandwidth values from when they are drawn (assumed to be empty[18]); that is for every established flow, the available bandwidth values of the links involved will be decremented accordingly and the new available bandwidth values will be used to compute path(s) for the next flow(s).

Since it is difficulty to show all the established flows for a single link, the program just indicates a used (non-empty) link with a different (red) colour. The actual flows that are existing in the network can be viewed using the "Established Links" option under the INFORMATION combo box.

The last important feature of the program is the information window which displays the necessary information on the program. The information can be displayed automatically as the program is used or by selecting the required information on the options under the INFORMATION combo box. Some of the later information includes:

- How to Draw/Delete/Move the nodes/links.
- How to change the required bandwidth values
- The Established/Blocked flows, etc.

Figure 3.10 below shows a bandwidth QoS program with arbitrary network and arbitrary network. The red links show the non-empty (used) links; the original link bandwidth values (when the links were empty) are written in black while the current available bandwidths are written in red. The green node is the destination node of the flow that has just been established. Some of the options under the INFORMATION combo box can also be seen.

[18] An empty link is one with no flows on it; same is for empty network

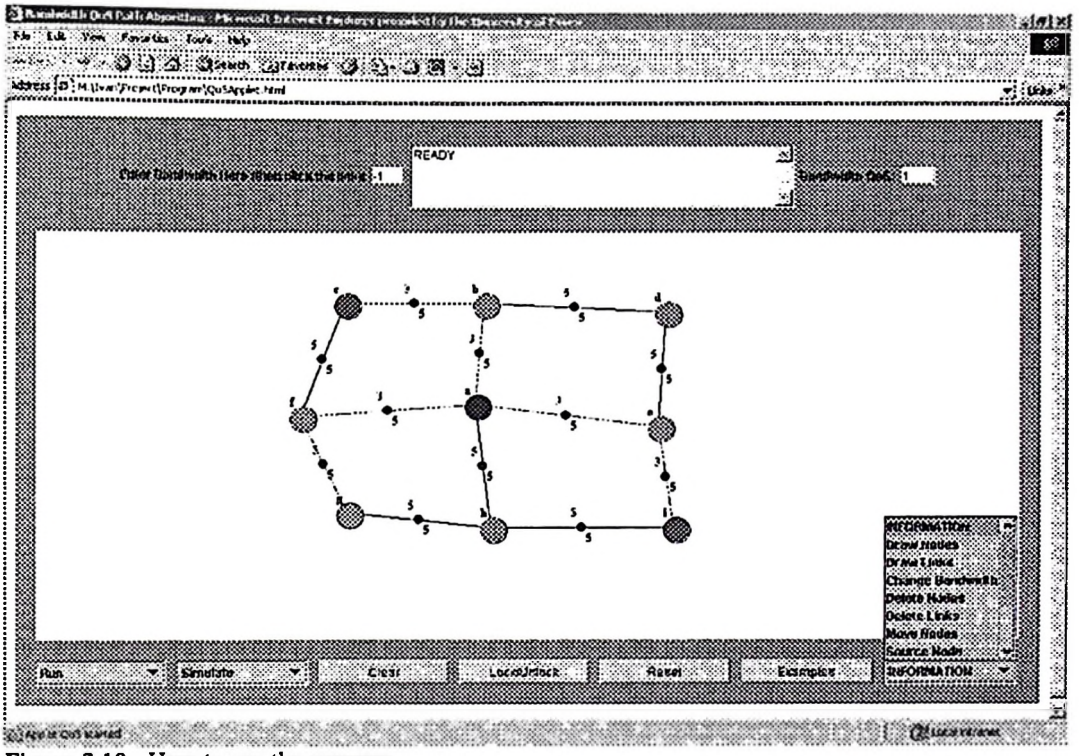


Figure 3.10: How to use the program.

3.4: Comparison of the Algorithms

As a final objective of the project, the developed algorithms: Dijkstra's and Bandwidth QoS algorithms were compared for different network circumstances. The aim is to see how efficient the new bandwidth QoS algorithm is as compared to the existing Dijkstra's algorithm.

The comparison model used in this project was to have different networks with single source node. For each node and for each algorithm, random flows are simulated from the source node to any of the other nodes (unicast[19] flows). Each time a flow is established the bandwidths of the links associated with it are decreased according to the bandwidth required by the flow. This process is done several times while recording the successfully established flows and the blocked ones. The whole simulation process is then done using the two different algorithms for different networks.

This comparison model is best done using so many different networks, but due to project duration and scope (which includes the design of the simulating program), two networks were sufficient to illustrate the comparison process. The two networks are described below.

Network A:

[19] A method by which a packet is sent to a single destination [10]

- Twenty five nodes arranged in a 5x5 square array with the source at the centre; grid point (3, 3).
- The nodes are linked in a square mesh; each node is linked to the nearest (if available) neighbour node on the left, right, top, or bottom of it.
- All the links have the same bandwidth value of 10 when the network is empty.
- All flows in this network were assumed to have a bandwidth requirement of *one*.

With this network the simulation was done ten times by *forth* (the maximum number of successfully established flows that can ever be) random flows and the results recorded for the two different algorithms. Figure 3.11 below shows the results of simulation with network A; source node in blue, last destination node in green and the previous destination nodes in red. Red links are the non-empty links.

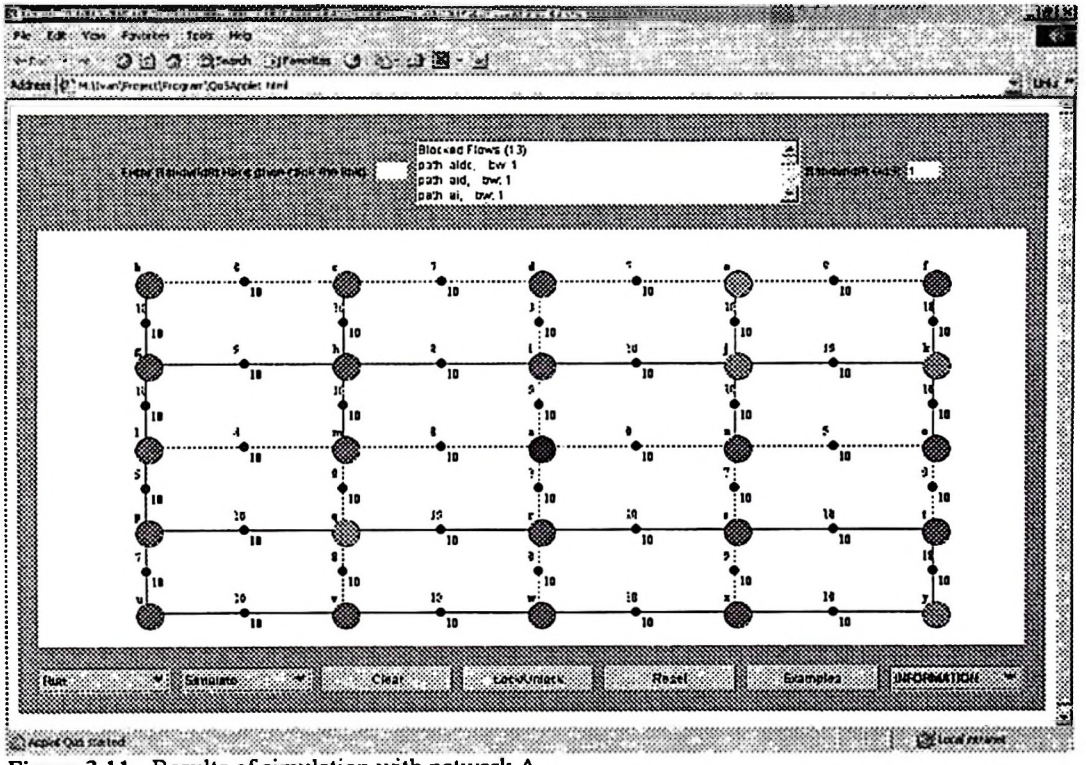


Figure 3.11: Results of simulation with network A.

Network B:

- Twenty two nodes linked in a three stage tree with the source at the root bearing three node children and the rest having two node children; 1 + (1x3) + (3x2) + (6x2) = 22.
- All the links have the same bandwidth value of 10 when the network is empty.
- All flows in this network were assumed to have a bandwidth requirement of *one*.

With this network the simulation was done ten times by *thirty* (the maximum number of successfully established flows that can ever be) random flows and the results recorded for the two different algorithms. Figure 3.12 below shows the results of simulation with network A

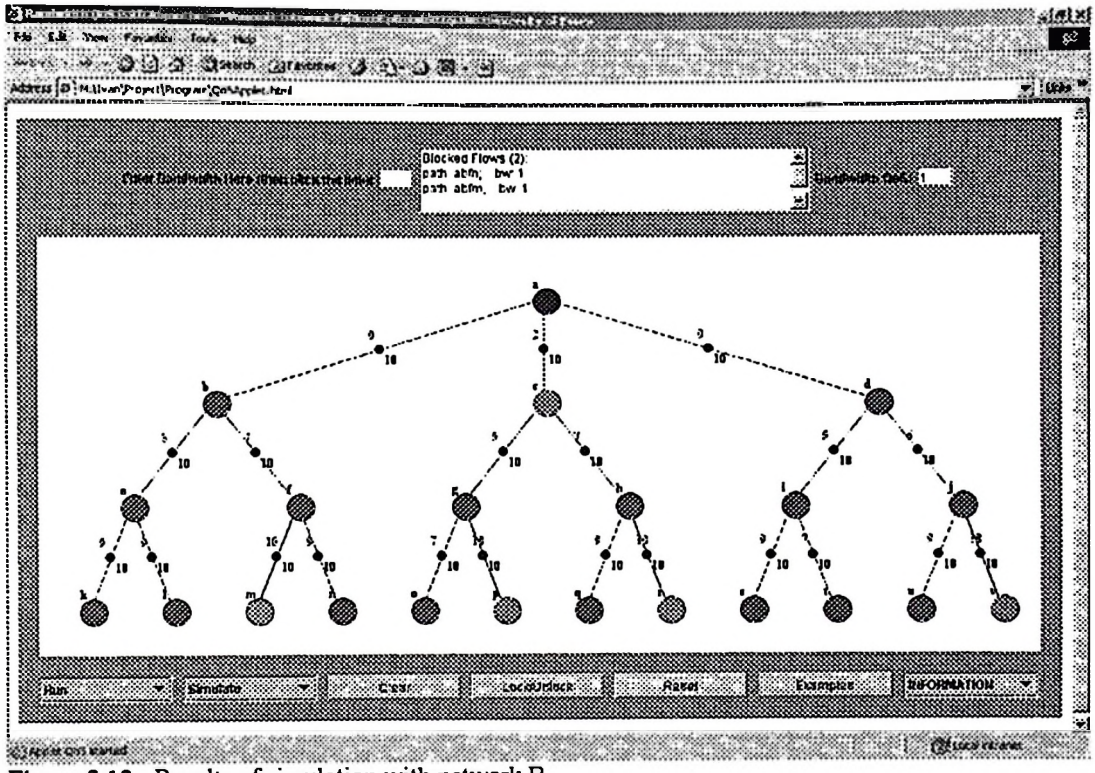


Figure 3.12: Results of simulation with network B.

4: Results

The tables in figures 4.1 and 4.2 below shows the results recorded after simulating network A using Dijkstra's algorithm and the bandwidth QoS algorithm respectively. Each simulation was done in 10 stages and for each stage *forty* flows to random destinations were being requested from the source node. Some of the flow requests were successfully established and others were blocked in the cases were the route (determined by the algorithm in use) to the destination does not meet the requested bandwidth (of *one* in this case). The first blocked request number is also noted.

After all of the *forty* flow-requests have been finished some of the links at the source (top, bottom, left and right) were completely used up (available bandwidth = 0) and some had some bandwidth leftovers. The tables also record the bandwidth leftovers of the links at the source.

Figure 4.1: A table to show the results of simulating network A with Dijkstra's algorithm.

				Source Link Bandwidth Leftover			
Simulation stage	Established flows	Blocked flows	First blocking	Left	Right	Top	Bottom
1	31	9	29	0	5	0	4
2	30	10	17	0	2	0	8
3	33	7	30	2	0	0	5
4	31	9	28	1	5	0	3
5	30	10	26	0	4	0	6
6	37	3	38	0	1	0	2
7	30	10	27	0	0	0	10
8	32	8	28	2	0	0	6
9	28	12	18	1	4	0	7
10	27	13	21	1	2	0	10
Average	30.9	9.1	26.2	0.7	2.3	0	6.1

Figure 4.2: A table to show the results of simulating network A with the bandwidth QoS algorithm.

				Source Link Bandwidth Left over			
Simulation stage	Established flows	Blocked flows	First blocking	Left	Right	Top	Bottom
1	40	0		0	0	0	0
2	40	0		0	0	0	0
3	40	0		0	0	0	0
4	40	0		0	0	0	0
5	40	0		0	0	0	0
6	40	0		0	0	0	0
7	40	0		0	0	0	0
8	40	0		0	0	0	0
9	40	0		0	0	0	0
10	40	0		0	0	0	0
Average	40	0	N/A	0	0	0	0

The tables in figures 4.3 and 4.4 below shows the results recorded after simulating network B using Dijkstra's algorithm and the bandwidth QoS algorithm respectively. Each simulation was done in 10 stages and for each stage *thirty* flows to random destinations were being requested from the source node. Some of the flow requests were successfully established and others were blocked in the cases were the route (determined by the algorithm in use) to the destination does not meet the requested bandwidth (of *one* in this case). The first blocked request number is also noted.

After all of the *thirty* flow-requests have been finished some of the links at the source (left, centre and right) were completely used up (available bandwidth = 0) and some had some bandwidth leftovers. The tables also record the bandwidth leftovers of the links at the source.

Figure 4.3: A table to show the results of simulating network B with Dijkstra's algorithm

				Source Link Bandwidth Left over		
Simulation stage	Established flows	Blocked flows	First blocking	Left	Centre	Right
1	25	5	25	0	0	5
2	27	3	25	0	1	2
3	28	2	29	1	1	0
4	25	5	21	0	2	3
5	27	3	25	0	0	3
6	24	6	20	0	0	6
7	25	5	25	5	0	0
8	30	0		0	0	0
9	29	1	30	1	0	0
10	27	3	26	3	0	0
Average	26.7	3.3	25.1	1	0.4	1.9

Figure 4.4: A table to show the results of simulating network B with the bandwidth QoS algorithm.

				Source Link Bandwidth Left over		
Simulation stage	Established flows	Blocked flows	First blocking	Left	Centre	Right
1	27	3	25	0	0	3
2	28	2	27	0	1	1
3	27	3	27	1	0	2
4	29	1	30	0	0	1
5	29	1	28	1	0	0
6	27	3	21	0	2	1
7	30	0		0	0	0
8	27	3	25	0	0	3
9	26	4	20	0	0	4
10	27	3	27	0	3	0
Average	27.7	2.3	25.6	0.2	0.6	1.5

5: Discussions, Conclusions and Recommendations

The Results

The simulation for network A using Dijkstra's algorithm shows that an average of 31 flows out of 40 were successfully established, which is equivalent to a throughput of 77.5%. It can also be seen that the earliest blocking occurred at the 17th request which means that the network is fully reliable (100% throughput) for the first 40% (16/40) flows.

The bandwidth leftovers depend on where the random destinations were (left, right, top, and bottom) in relation to the source. Due to the fact that Dijkstra's algorithm has no sense of load-balancing[20], some of the links (the top link is an example in this case) are always considered for use and hence get used up quickly.

The results obtained using the bandwidth QoS algorithm gives 100% throughput all the times. This is so because the algorithm do consider several paths to the destination and picks up the one with least number of hops and highest bottleneck bandwidth unlike the Dijkstra's algorithm that always goes for the shortest (least number of hops) path.

The results for network B do not show significant difference between the two algorithms. All have average throughput at around 90% with the network fully reliable for the first 63% flows. There is also no significance difference on the utilization of the links despite the fact that the bandwidth QoS algorithm has a sense of load-balancing while Dijkstra's algorithm hasn't.

We obtain different comparison results for the two networks because the two networks are very different. Network A has all nodes linked to at least two other nodes and most of the nodes are linked to four other nodes while in network B most of the nodes are linked to one or two other nodes. This makes network A more meshy which allows many paths to the destination unlike network B that has a single path for every destination. With network B the bandwidth QoS algorithm will have only one path to choose and so it will end up operating like the Dijkstra's algorithm. So the bandwidth QoS algorithm operates more efficiently in more meshy networks than in less meshy ones.

A better way to obtain the comparison results could be done by first determine a network that best models the internet and use it for simulations to obtain the results. Alternatively, the simulations could be done with so many different networks and the results combined and compared.

It should be noted that these results are obtained under the non-realistic (in practise) assumptions that all of the flows are unicast originating from the same source and all have the same bandwidth requirements. Slight modifications to the program are necessary to make it handle more realistic networks with multicast[21] flows of random bandwidth requirements from random sources and hence obtain more genuine results.

---

[20] Distributing flows according to the available bandwidth in the links or according to number of flows that are already available in the links

[21] A technique that allows copies of a single packet to be passed to a selected subnet of all possible destinations.

The bandwidth QoS Algorithm

The bandwidth QoS algorithm studied in this project assumes that each router maintains an updated database of the network topology, including current state (e.g. available bandwidth) of each link [1]. This is possible only if the routers frequently advertise link metric updates i.e. whenever a change in the network occurs. On the other hand, this frequent advertisement will consume much of the link bandwidths that may be a scarce resource. In general, there is a trade-off between the efficient bandwidth usage (by fewer advertisements) and the accuracy of the network state information that the path selection algorithm depends on [1].

It is suggested in [1] that the link state advertisements being triggered only when there is a significant change in the value of the metrics (since the last advertisement) as a practical compromise of the trade-off.

To minimise impacts to the existing OSPF protocol, the algorithm is limited to considering only bandwidth requirements as an additional QoS metric. Other QoS metrics like reliability, delay and jitter are not highly considered. To deal with delays [1] propose the use of a technique that would eliminate from the network all links with high propagation delay, e.g. satellite links, before invoking the algorithm. Alternatively, different algorithms could be used depending on the QoS requirements expressed by an incoming request. All these two proposals can be implemented at a cost of increase in the routers' processing overhead.

In general terms the algorithm performs the path computation as follows. It finds the route with minimum hop count that meets the bandwidth QoS. If there is more than one such route, the algorithm takes the route that has highest bottleneck link bandwidth. The idea is to keep the bandwidth leftovers of the links as high as possible, so if there are two paths to a destination, one with a bandwidth of 10 and the other with bandwidth of 20, the algorithm will use the later path to establish a flow of bandwidth requirement of say 5. Hence the bandwidth leftovers will be 10 and 15.

It can be argued that there is a significant advantage if the algorithm would consider the route with lowest bottleneck bandwidth. In the previous example, using lowest bandwidth route will make the route with bandwidth 10 be used and hence the bandwidth leftover will be 5 and 20. In this case, the network will be able to route all flows of bandwidth requirement up to 20 where as with the use of highest bandwidth route the network will be able to route flows of bandwidth requirement only up to 15.

With this significant advantage, it is recommended that further studies of the algorithm be done on whether the algorithm should use the highest or lowest bandwidth route. It may also worth to use the algorithm in both ways depending on the network condition.

References

[1] "QoS Routing Mechanisms and OSPF Extensions"; RFC 2676, August 1999. G. Apostolopoulos, D. Williams, S. Kamat, R. Guerin, A.Orda, T. Przygienda.

[2] Java for Engineers and Scientists; Stephen J. Chapman

[3] Computer Networks (4th Edition); A. S. Tannenbaum.

[4] Routing in the Internet (2nd Edition); C. Huitema.

[5] EE 904 Lecture notes; Dr. S. Monaghan.

[6] "A Framework for QoS-based Routing in the Internet"; RFC 2386, August 1998. E. Crawley, R. Nair, B. Rajagopalan, H. Sandick.

[7] Numerical Recipes in C (2nd Edition); W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery.

[8] EE 935 Lecture notes; Dr. S. Monaghan.

[9] Java: How to Program; H. M. Deitel, P. J. Deitel.

[10] Internetworking with TCP/IP; D. E. Comer

[11] http://carbon.cudenver.edu/~hgreenbe/sessions/dijkstra/DijkstraApplet.html

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.awt.Graphics;
import java.util.*;
/*the area where the network will be drawn*/
public class PlayGround extends Panel implements MouseListener, MouseMotionListener
{
        final int MAXNODES = 27;
         final int NODESIZE = 26;
        final int SOURCE = 1;

        int[] leastWeight = new int[MAXNODES];
        String[] path = new String[MAXNODES];

        boolean Locked = false,  runDij = false, runQoS = false, Ctrl = false, set = false,
                QoSMet = false;

        Node[] node = new Node[MAXNODES];
        Link[][] link = new Link[MAXNODES][MAXNODES];
        Ball[][] ball = new Ball[MAXNODES][MAXNODES];
        tab_entry[][] TT = new tab_entry[MAXNODES+1][MAXNODES+1];// topology table

        int xValue = -30 , yValue = -30, nodecount = 1;
        int tempStart, tempEnd,
                dest,     //destination node
                blockCount,//number of blocked flows
                flowNum,          //number of established flows
                req;  // number of requests

        String flows = "",//established flows
                  blocks = "", //blocked flows
                  blockingAt = "";          //req # at which blocking occurs

        QoS parent;

        /*Constructor*/
        public PlayGround (QoS myparent)
        {
                parent = myparent;
                addMouseListener (this);
                addMouseMotionListener (this);
                setBackground(Color.WHITE);

                for (int i=1;i<MAXNODES;i++)
                        for (int j=1;j<MAXNODES;j++)
                                link[i][j] = new Link();
        }

        /*lock screen */
        public void lock()
        {
                Locked=true;
        }

  /*unlock screen */
        public void unlock()
        {
        Locked=false;
  }
```

APPENDICES: The Program Codes

```
import java.awt.*;          import java.awt.event.*;
import javax.swing.*;       import java.awt.Graphics;
import java.util.*;
/*the area where the network will be drawn*/
public class PlayGround extends Panel implements MouseListener, MouseMotionListener
{
        final int MAXNODES = 27;
         final int NODESIZE = 26;
        final int SOURCE = 1;

        int[] leastWeight = new int[MAXNODES];
        String[] path = new String[MAXNODES];

        boolean Locked = false,  runDij = false, runQoS = false, Ctrl = false, set = false,
                QoSMet = false;

        Node[] node = new Node[MAXNODES];
        Link[][] link = new Link[MAXNODES][MAXNODES];
        Ball[][] ball = new Ball[MAXNODES][MAXNODES];
        tab_entry[][] TT = new tab_entry[MAXNODES+1][MAXNODES+1];// topology table

        int xValue = -30 , yValue = -30, nodecount = 1;
        int tempStart, tempEnd,
                dest,     //destination node
                blockCount,//number of blocked flows
                flowNum,          //number of established flows
                req;  // number of requests

        String flows = "", //established flows
                  blocks = "", //blocked flows
                  blockingAt = "";          //req # at which blocking occurs

        QoS parent;

        /*Constructor*/
        public PlayGround (QoS myparent)
        {
                parent = myparent;
                addMouseListener (this);
                addMouseMotionListener (this);
                setBackground(Color.WHITE);

                for (int i=1;i<MAXNODES;i++)
                        for (int j=1;j<MAXNODES;j++)
                                link[i][j] = new Link();
        }

        /*lock screen */
        public void lock()
        {
                Locked=true;
        }

   /*unlock screen */
        public void unlock()
        {
        Locked=false;
   }
```

```
/*Invoked when a mouse button has been pressed on a component.*/
public void mousePressed (MouseEvent evt)
{
  if (!Locked)
        {

        //select the destination node by right-click
        if (evt.getButton()==MouseEvent.BUTTON3)
        {
            for (int j=2; j<nodecount;j++)
                if (length(evt.getX(),evt.getY(),node[j].xPos+13,node[j].yPos+13)<13)
                        {
                                dest = j;
                                repaint();
                        }
        }
        else
        //select the node to delete using the centre
        if (evt.getButton()==MouseEvent.BUTTON2)
        {
            for (int j=2; j<nodecount;j++)
                if (length(evt.getX(),evt.getY(),node[j].xPos+13,node[j].yPos+13)<13)
                        {
                                node[j] = new Node();
                                for (int i=1; i<nodecount; i++)
                                {
                                        link[i][j].bwidth = -1;
                                        link[j][i].bwidth = -1;
                                }
                                nodecount--;
                                repaint();
                        }
        }
        else
        //if (!Locked)
{
        boolean tooNear = false;
        tempStart = 20; tempEnd = 20;
        //int m , c;

        for (int i=1; i<nodecount;i++)
        {
                if (length(evt.getX(),evt.getY(),node[i].xPos,node[i].yPos)<50)
                        tooNear = true;
                //else
                        for (int j=i+1;j<nodecount;j++)
                        if (link[i][j].exist)
                        {
                                link[i][j].ballX =(node[i].xPos+node[j].xPos)/2+5;
                                link[i][j].ballY = (node[i].yPos+node[j].yPos)/2+5;
                                if (length(evt.getX(),evt.getY(),link[i][j].ballX,
                                                link[i][j].ballY)<50)
                                        tooNear = true;
                                if (length(evt.getX(),evt.getY(),link[i][j].ballX+5,
                                                link[i][j].ballY+5)<5)
                                  if(!set)
                                {
                                    link[i][j].bwidth = link[j][i].bwidth = link[i][j].bw =
                        link[j][i].bw = Integer.parseInt(parent.display.text1.getText());
```

```
/*Invoked when a mouse button has been pressed on a component.*/
public void mousePressed (MouseEvent evt)
{
  if (!Locked)
        {

        //select the destination node by right-click
        if (evt.getButton()==MouseEvent.BUTTON3)
        {
            for (int j=2; j<nodecount;j++)
                if (length(evt.getX(),evt.getY(),node[j].xPos+13,node[j].yPos+13)<13)
                        {
                                dest = j;
                                repaint();
                        }
        }
        else
        //select the node to delete using the centre
        if (evt.getButton()==MouseEvent.BUTTON2)
        {
            for (int j=2; j<nodecount;j++)
                if (length(evt.getX(),evt.getY(),node[j].xPos+13,node[j].yPos+13)<13)
                        {
                                node[j] = new Node();
                                for (int i=1; i<nodecount; i++)
                                {
                                        link[i][j].bwidth = -1;
                                        link[j][i].bwidth = -1;
                                }
                                nodecount--;
                                repaint();
                        }
        }
        else
        //if (!Locked)
 {
        boolean tooNear = false;
        tempStart = 20; tempEnd = 20;
        //int m , c;

        for (int i=1; i<nodecount;i++)
        {
                if (length(evt.getX(),evt.getY(),node[i].xPos,node[i].yPos)<50)
                        tooNear = true;
                //else
                        for (int j=i+1;j<nodecount;j++)
                        if (link[i][j].exist)
                        {
                                link[i][j].ballX =(node[i].xPos+node[j].xPos)/2+5;
                                link[i][j].ballY = (node[i].yPos+node[j].yPos)/2+5;
                                if (length(evt.getX(),evt.getY(),link[i][j].ballX,
                                                link[i][j].ballY)<50)
                                        tooNear = true;
                                if (length(evt.getX(),evt.getY(),link[i][j].ballX+5,
                                                link[i][j].ballY+5)<5)
                                  if(!set)
                                {
                                    link[i][j].bwidth = link[j][i].bwidth = link[i][j].bw =
                        link[j][i].bw = Integer.parseInt(parent.display.text1.getText());
```

```
                                repaint();
                            }//if
                        }//if
                if (length(evt.getX(),evt.getY(),node[i].xPos+13,node[i].yPos+13)<13)
                        tempStart = i;
        }//for

        if (!tooNear)
        {
                node[nodecount] = new Node();
                node[nodecount].xPos = evt.getX();
                node[nodecount].yPos = evt.getY();
                repaint();
                nodecount++;
        }
        }
        }
}

/*Invoked when a mouse button is pressed on a component and then dragged.*/
public void mouseDragged (MouseEvent evt)
{
        if (!Locked)
        for (int j=1; j<nodecount;j++)
        {
                if (length(evt.getX(),evt.getY(),node[j].xPos+13,node[j].yPos+13)
                        <13)
                {
                        tempEnd = j;
                        link[tempStart][tempEnd].exist = true;
                        link[tempEnd][tempStart].exist = true;
                        link[tempStart][tempEnd] = link[tempEnd][tempStart];
                }
        }
        repaint();

}

/*Invoked when the mouse cursor has been moved onto a component but
no buttons have been pushed.  */
public void mouseMoved (MouseEvent evt)
{       }

/*Invoked when the mouse button has been clicked (pressed and released)
on a component.*/
public void mouseClicked (MouseEvent evt)
{       }

/*Invoked when a mouse button has been released on a component.*/
public void mouseReleased (MouseEvent evt)
{       }

/*Invoked when the mouse enters a component.*/
public void mouseEntered(MouseEvent evt)
{       }

/*Invoked when the mouse exits a component.*/
public void mouseExited (MouseEvent evt)
{       }
```

```
/*Paints the playgroung with nodes, links, etc*/
public void paint (Graphics g)
{
        //Draw lines
        for (int i=1; i<nodecount; i++)
        {
         for (int j=i+1; j<nodecount;j++)
                {
                 if (link[i][j].exist)
                        {
                          if (link[i][j].bwidth>=0)
                          {
                                if(link[i][j].bwidth==link[i][j].bw)//if link is unused
                                {
                                  //draw it in black
                                        g.setColor(Color.BLACK);
                                        g.setFont(new Font ("Serif",Font.BOLD,12));
                                        g.drawLine(node[i].xPos+10, node[i].yPos+10,
                                                node[j].xPos+10,node[j].yPos+10);
                                }else     //if the link is used
                                {
                                        //draw it in red
                                        g.setColor(Color.RED);
                                        g.setFont(new Font ("Serif",Font.BOLD,12));
                                        g.drawLine(node[i].xPos+10, node[i].yPos+10,
                                                node[j].xPos+10,node[j].yPos+10);
                                }
                                //draw the balls at the centre of the link and
                                //write bandwidth
                                g.setColor(Color.BLACK);
                                link[i][j].ballX =(node[i].xPos+node[j].xPos)/2+5;
                                link[i][j].ballY = (node[i].yPos+node[j].yPos)/2+5;
                                g.drawString(""+link[i][j].bw,link[i][j].ballX+10,
                                                        link[i][j].ballY+20);
                                g.fillOval(link[i][j].ballX, link[i][j].ballY, 10, 10);
                                g.setColor(Color.RED);
                                g.drawString(""+link[i][j].bwidth,link[i][j].ballX-5,
                                                        link[i][j].ballY-5);
                          }
                          //if link doesnt exist
                          else link[i][j] = link[j][i] = new Link();
                        }//if
                }//for
  }//for

        g.setFont(new Font ("Serif",Font.BOLD,12));

        //draw first(source) node in blue
        g.setColor (Color.BLUE);
        g.fillOval(node[1].xPos, node[1].yPos, NODESIZE, NODESIZE);
        g.setColor(Color.BLACK);
        g.drawOval(node[1].xPos, node[1].yPos, NODESIZE, NODESIZE);
        g.drawString(intToString(1), node[1].xPos, node[1].yPos);

        //draw other nodes
        for (int i=2; i<nodecount; i++)
        //if (i!=dest)
        {
                g.drawOval(node[i].xPos, node[i].yPos, NODESIZE, NODESIZE);
                g.drawString(intToString(i), node[i].xPos, node[i].yPos);
```

```
                g.setColor(Color.GRAY);
                g.fillOval(node[i].xPos, node[i].yPos, NODESIZE, NODESIZE);
                g.setColor(Color.BLACK);
        }

        //draw destination nodes
        for (int i=1; i<nodecount;i++)
                if (node[i].name == 'd')
                {
                        g.setColor (Color.RED);
                        g.fillOval(node[i].xPos, node[i].yPos, NODESIZE, NODESIZE);
                        g.setColor(Color.BLACK);
                        g.drawOval(node[i].xPos, node[i].yPos, NODESIZE, NODESIZE);
                        g.drawString(intToString(i), node[i].xPos, node[i].yPos);
                }


        //draw curent destination node is green
        g.setColor (Color.GREEN);
        g.fillOval(node[dest].xPos, node[dest].yPos, NODESIZE, NODESIZE);
        g.setColor(Color.BLACK);
        g.drawOval(node[dest].xPos, node[dest].yPos, NODESIZE, NODESIZE);
        g.drawString(intToString(dest), node[dest].xPos, node[dest].yPos);

}

/*converts an integer into a character; eg: 1='a', 2='b', etc*/
public String intToString (int num)
{
        String str = null;
        switch (num)
        {
        case 1:  str = "a"; break;  case 2: str = "b"; break;
        case 3: str = "c"; break;   case 4: str = "d"; break;
        case 5: str = "e"; break;   case 6: str = "f"; break;
        case 7:  str = "g"; break;  case 8: str = "h"; break;
        case 9:  str = "i"; break;  case 10: str = "j"; break;
        case 11: str = "k"; break;  case 12: str = "l"; break;
        case 13: str = "m"; break;  case 14: str = "n"; break;
        case 15: str = "o"; break;  case 16: str = "p"; break;
        case 17: str = "q"; break;  case 18: str = "r"; break;
        case 19: str = "s"; break;  case 20: str = "t"; break;
        case 21: str = "u"; break;  case 22: str = "v"; break;
        case 23: str = "w"; break;  case 24: str = "x"; break;
        case 25: str = "y"; break;  default: str = ""; break;
        }
        return str;
}

/*converts a character into an integer; a=1, b=2, etc*/
public int charToint (char ch)
{
        int num = 0;
        switch (ch)
        {
        case 'a': num = 1; break;   case 'b': num = 2; break;
        case 'c': num = 3; break;   case 'd': num = 4; break;
        case 'e': num = 5; break;   case 'f': num = 6; break;
        case 'g': num = 7; break;   case 'h': num = 8; break;
        case 'i': num = 9; break;   case 'j': num = 10; break;
```

```
            case 'k': num = 11; break;  case 'l': num = 12; break;
            case 'm': num = 13; break;  case 'n': num = 14; break;
            case 'o': num = 15; break;  case 'p': num = 16; break;
            case 'q': num = 17; break;  case 'r': num = 18; break;
            case 's': num = 19; break;  case 't': num = 20; break;
            case 'u': num = 21; break;  case 'v': num = 22; break;
            case 'w': num = 23; break;  case 'x': num = 24; break;
            case 'y': num = 25; break;  default: num = 0; break;
            }
            return num;
    }

    /*returns a length between two points*/
    public double length (int x1,int y1,int x2,int y2)
    {
            return Math.sqrt(Math.pow((x1-x2),2)+Math.pow((y1-y2),2));
    }

    /*clears the network from the screem.*/
    public void clear()
    {
            for(int i=1; i<MAXNODES; i++)
            {
                    node[i]=new Node();
                    nodecount = 1;
                    leastWeight = new int[MAXNODES];
                    path = new String[MAXNODES];
                    for (int j=1;j<MAXNODES; j++)
                            link[i][j] = new Link();
            }
    repaint();
}

    /*Runs Dijkstra's algorithm to produce shortest distaces/routes to some  specified nodes*/
    public void runDijkstra ()
    {
            int st,nd;
            runDij = true; set =true; //Locked = true;

            //set initial least weights and paths form the source
            for (int i=1; i< nodecount; i++)
            {
                    leastWeight[i] = 1000000; // set to infinity
                    path[i] = "";       // clear path
            }
            leastWeight[SOURCE] = 0;        //set least weight from itself to zero

            //A List of unvisited nodes
            LinkedList destNode = new LinkedList();

            //Insert nodes in to the List with source as the first node
            destNode.addFirst(new Integer(SOURCE));
            for (int j=1; j<nodecount; j++)
            {
                    if (j != SOURCE)
                    {
                            destNode.addLast(new Integer(j));
                    }
            }
```

```
                Object curr = null;
                int current, adjacent;
                while(!destNode.isEmpty())
                {
                        curr = destNode.removeFirst();
                        current = Integer.parseInt(curr.toString());
                        path[current] = path[current] + intToString(current) ;//+ " ";

                        for(int i=1; i<nodecount; i++)
                        {
                                // look for adjacent vertexes
                                if ( (link[current][i].exist)&& (current!=i))
                                {
                                        adjacent = i;
                                        if (leastWeight[adjacent] >
                                                leastWeight[current]+
link[current][adjacent].weight)
                                        {

                                                // update the distance from source
                                                leastWeight[adjacent] = leastWeight[current] +

        link[current][adjacent].weight;
                                                path[adjacent] = path[current];       // update path
                                        }//if
                                }//if
                        }//for
                        if (!destNode.isEmpty()) reorder(destNode,leastWeight);// reorder
                }
                //read the bandwidth QoS required:
                int bw_QoS = Integer.parseInt(parent.display.text2.getText());
                boolean QoS_Met = true;  // a flag
                //check if all links meet QoS
                for (int j=0;j<path[dest].length()-1;j++)
                {
                        st = charToint(path[dest].charAt(j));
                        nd = charToint(path[dest].charAt(j+1));
                        if(link[st][nd].bwidth < bw_QoS)   //if QoS not met by a link
                        {
                                QoS_Met = false;
                                j = path[dest].length();      //stop looping
                        }
                }

                if((bw_QoS>0)&&QoS_Met)
                {
                        QoSMet = true;
                        req++;
                        parent.display.area1.setText("A new flow established" +
                                "\nbandwidth = "+ bw_QoS + "\npath = "+path[dest]);
                        //flows = flows + "path: "+path[dest]+";    bw: "+bw_QoS+"\n";
                        flows = flows + path[dest]+"; ";
                        flowNum++;
                        node[charToint(path[dest].charAt(path[dest].length()-1))].name = 'd';
                        if (path[dest].length()>1)
                        {
                                for (int j=0;j<path[dest].length()-1;j++)
                                {
                                        st = charToint(path[dest].charAt(j));
                                        nd = charToint(path[dest].charAt(j+1));
```

```
                                        //decrement link bandwidth
                                        link[st][nd].bwidth = link[st][nd].bwidth - bw_QoS;
                                        link[nd][st].bwidth = link[nd][st].bwidth - bw_QoS;
                                }
                        }
                }
                else
                {
                        parent.display.area1.setText("BLOCKED! path don't meet QoS"+

"\n\t\tLOCKED! Press Unlock");
                        blockCount++,
                        req++;
                        blockingAt = blockingAt + " " + req;
                        //blocks = blocks +"path: "+path[dest]+";    bw: "+bw_QoS+"\n";
                        blocks = blocks + path[dest]+"; ";
                }
        }

        /*Runs the bandwidth QoS algorithm to produce bandwidth/routes to some specified nodes*/
        public void runBwQoS ()
        {
                Object[] sPrev;
                int st,nd;
                runQoS = true; set = true; QoSMet = false; //Locked = true;

                for (int i=1; i<nodecount; i++)       // initialization
                {
                        TT[i][0] = new tab_entry();
                        TT[i][1] = new tab_entry();
                        TT[i][1].path = "" + intToString(SOURCE);
                        TT[i][0].path = "" + intToString(SOURCE);

                }

                TT[SOURCE][0].bw       = 1000000;

                //reset s_prev
                LinkedList s_prev = new LinkedList();

                for (int n=1; n<nodecount; n++)     // for all nodes n
                {
                        if (link[SOURCE][n].exist)          // that are neighbour to source
                        {
                                //top_table[n][1].bw = bwidth[source][n];
                                TT[n][1].bw = max(TT[n][1].bw, link[SOURCE][n].bwidth);
                                Integer N = new Integer(n);           // convert to object
                                if (TT[n][1].bw == link[SOURCE][n].bwidth)
                                {
                                        TT[n][1].neighbor = N;
                                        TT[n][1].path = TT[n][1].path + intToString(n);
                                }
                                //add a router in S_prev if not already included in S_prev
                                if (!s_prev.contains(N)) s_prev.addLast(N);
                        }
                }
                for (int h=2; h<nodecount-1; h++)  //consider all number of hops
                {
                        // reset s_new;
                        LinkedList s_new = new LinkedList();
```

```
        for (int m=1; m<nodecount; m++)  //for all vertices m
        {
                TT[m][h] = new tab_entry();
                TT[m][h].bw = TT[m][h-1].bw;
                TT[m][h].neighbor = TT[m][h-1].neighbor;
                TT[m][h].path = TT[m][h-1].path;
        }
        sPrev = s_prev.toArray();// put data from list to array
        for (int i=0; i<sPrev.length; i++)                    // for all vertices
        {
                int n = Integer.parseInt(sPrev[i].toString());// n in s_prev
                for (int m=1; m<nodecount; m++)  // for all possible
                {
                        if (link[n][m].exist)          // edges (n,m)
                        {
                                if (min(TT[n][h-1].bw, link[n][m].bwidth) >
                                TT[m][h].bw)
                                {
                                   TT[m][h].bw = min(TT[n][h-1].bw,
                                   link[n][m].bwidth);
                                   TT[m][h].neighbor = TT[n][h-1].neighbor;
                                   TT[m][h].path = TT[n][h-1].path  +
                                   intToString(m);
                                   Integer M = new Integer(m);// convert to
                                   object
                                    if (!s_new.contains(M))
                                                s_new.addLast(M); //routers
                                                are added to s_new
                                }//if
                        }//if
                }//for
        }//for

        s_prev = s_new;

        if (s_prev == null)
                h = nodecount-1; // if no changes then exit
}

//read the bandwidth QoS required:
int bw_QoS = Integer.parseInt(parent.display.text2.getText());

//traversing the router table:
for (int j=1;j<nodecount-2;j++)// for all possible hopcounts
        if (TT[dest][j].path.length()>1)
        {
        if ((bw_QoS>0)&&(TT[dest][j].bw>= bw_QoS))// if bw meets QoS
                {
                        QoSMet = true;
                        flowNum++;
                        req++;
                        //write bw and path on the display:
                        parent.display.area1.setText("A new flow established"+
                                "\nbandwidth = "+ bw_QoS + "\npath = " +
                                                TT[dest][j].path );

                        //flows = flows + "path: "+TT[dest][j].path + ";    bw: "
                        //        +bw_QoS+"\n";
```

```
                                        flows = flows + TT[dest][j].path + "; ";
                                        node[charToint(TT[dest][j].path.charAt
                                                (TT[dest][j].path.length()-1))].name = 'd';

                                        for (int k=0;k<TT[dest][j].path.length()-1;k++)
                                        {
                                                st = charToint(TT[dest][j].path.charAt(k));
                                                nd = charToint(TT[dest][j].path.charAt(k+1));

                                                //decrement link bandwidth:
                                                link[st][nd].bwidth = link[st][nd].bwidth -
                                                bw_QoS;
                                                link[nd][st].bwidth = link[nd][st].bwidth -
                                                bw_QoS;
                                        }

                                        //break out the loop(write only the first one):
                                        j=nodecount;
                                }
                        }
                if (!QoSMet)
                {
                        parent.display.area1.setText("BLOCKED! path don't meet QoS"+
                                "\n\t\tLOCKED! Press Unlock");
                        blockCount++;
                        req++;
                        blockingAt = blockingAt + " " + req;
                        //blocks = blocks +"path: "+TT[dest][j].path+";    bw: "+bw_QoS+"\n";
                        blocks = blocks + intToString(dest) + "; ";
                        QoSMet = false;
                }
        }// end runBwQoS

        /*Orders values in a LinkedList list according the their corresponding
        values in an array dist*/
        public synchronized void reorder(LinkedList list,int[] dist)
{
          int[] tempArray = new int[list.size()];
          int i = 0;

          while ( !list.isEmpty() )
          {
                  tempArray[i] = Integer.parseInt(list.removeFirst().toString());
                  i = i + 1;
          }

          sort(tempArray, dist);

          //Integer J = null;
          for (int j=0; j<tempArray.length; j++)
          {
                  //J = new Integer(tempArray[j]);
                  list.addLast(new Integer(tempArray[j]));
          }

}
```

```
        /*sorts values in array arr according to their corresponding values
        in array brr*/
public synchronized void sort (int[] arr,int[] brr )
{
        int i, a, j, b;

        for (j=1; j<arr.length; j++) //Pick out each element in turn
        {
                a=arr[j]; b=brr[a];             i=j-1;
                while(i>=0 && brr[arr[i]]>b) //Find a place to put
                {
                        arr[i+1] = arr[i];  i--;
                }
                arr[i+1] = a;                                           // Put it
        }

}

/*returns the minimun between the two integers a and b*/
public static int min(int a, int b)
        {
                int min;
                min = a;
                if (b < a)min = b;
                return min;
        }

/*returns the maximum between the two integers a and b*/
public static int max(int a, int b)
        {
                int max;
                max = a;
                if (b > a)max = b;
                return max;
        }

        /*resets the network to an empty network*/
public void reset()
        {
                dest = 0;
                for (int i=1; i<nodecount; i++)
                {
                 node[i].name ='?';
                 for (int j=i+1; j<nodecount;j++)
                  if (link[i][j].exist)  //||link[j][i].exist)
                  {
                          link[i][j].bwidth = link[i][j].bw = 5;          repaint();
                  }
                 }
        }

/*one example of the possible networks: 5x5 square array of nodes*/
        public void example1()
        {
 // draw a simulation network
        int wth, hgt;       clear();  nodecount=26;

        //draw nodes:
        wth=this.getSize().width/10;        hgt=this.getSize().height/10;
```

```
        node[2]=new Node(wth, hgt);     node[3]=new Node(3*wth, hgt);
        node[4]=new Node(5*wth, hgt);   node[5]=new Node(7*wth, hgt);
        node[6]=new Node(9*wth, hgt);     node[7]=new Node(wth, 3*hgt);
        node[8]=new Node(3*wth, 3*hgt); node[9]=new Node(5*wth, 3*hgt);
        node[10]=new Node(7*wth, 3*hgt);          node[11]=new Node(9*wth, 3*hgt);
        node[12]=new Node(wth, 5*hgt);   node[13]=new Node(3*wth, 5*hgt);
        node[1]=new Node(5*wth, 5*hgt); node[14]=new Node(7*wth, 5*hgt);
        node[15]=new Node(9*wth, 5*hgt);          node[16]=new Node(wth, 7*hgt);
        node[17]=new Node(3*wth, 7*hgt);          node[18]=new Node(5*wth, 7*hgt);
        node[19]=new Node(7*wth, 7*hgt);          node[20]=new Node(9*wth, 7*hgt);
        node[21]=new Node(wth, 9*hgt);   node[22]=new Node(3*wth, 9*hgt);
        node[23]=new Node(5*wth, 9*hgt);   node[24]=new Node(7*wth, 9*hgt);
        node[25]=new Node(9*wth, 9*hgt);

        //draw horizontal links:
        link[2][3].exist = link[3][4].exist = link[4][5].exist =
        link[5][6].exist = link[7][8].exist = link[8][9].exist =
        link[9][10] exist =          link[10][11].exist = link[12][13].exist =
        link[1][13].exist = link[1][14].exist = link[14][15].exist =
        link[16][17].exist = link[17][18].exist = link[18][19].exist =
        link[19][20].exist = link[21][22].exist = link[22][23].exist =
        link[23][24].exist = link[24][25].exist = true;
        //draw vertical links:
        link[2][7].exist= link[7][12].exist = link[12][16].exist =
        link[16][21].exist = link[3][8].exist = link[8][13].exist =
        link[13][17].exist = link[17][22].exist = link[4][9] exist =
        link[1][9].exist = link[1][18].exist = link[18][23].exist =
        link[5][10].exist = link[10][14].exist = link[14][19].exist =
        link[19][24].exist = link[6][11].exist = link[11][15].exist =
        link[15][20].exist = link[20][25] exist = true;

        for (int i=1; i<nodecount; i++)
                for (int j=i+1; j<nodecount; j++)
                {
                        if (link[i][j].exist)
                                link[j][i].exist = true;
                        if (link[j][i].exist)
                                link[i][j].exist = true;
                }
        repaint();
}

        /*one example of the possible networks: 3-stage tree*/
        public void example2()
        {
// draw a simulation network
        int wth, hgt;       clear();   nodecount=23;

        //draw nodes:
        wth = this.getSize().width/24;        hgt = this.getSize().height/8;

        node[1] = new Node(12*wth, hgt); //source node at the root
        //first stage nodes:
        node[2] = new Node(4*wth, 3*hgt);          node[3] = new Node(12*wth, 3*hgt);
        node[4] = new Node(20*wth, 3*hgt);
        //second stage nodes:
        node[5] = new Node(2*wth, 5*hgt);          node[6] = new Node(6*wth, 5*hgt);
        node[7] = new Node(10*wth, 5*hgt);  node[8] = new Node(14*wth, 5*hgt);
        node[9] = new Node(18*wth, 5*hgt);  node[10] = new Node(22*wth, 5*hgt);
        //third stage nodes:
```

```
        node[11] = new Node(wth, 7*hgt); node[12] = new Node(3*wth, 7*hgt);
        node[13] = new Node(5*wth, 7*hgt); node[14] = new Node(7*wth, 7*hgt);
        node[15] = new Node(9*wth, 7*hgt);          node[16] = new Node(11*wth, 7*hgt);
        node[17] = new Node(13*wth, 7*hgt);         node[18] = new Node(15*wth, 7*hgt);
        node[19] = new Node(17*wth, 7*hgt); node[20] = new Node(19*wth, 7*hgt);
        node[21] = new Node(21*wth, 7*hgt); node[22] = new Node(23*wth, 7*hgt);
        //draw 1st stage links:
        link[1][2].exist = link[1][3].exist = link[1][4].exist =
        //draw 2nd stage links:
        link[2][5].exist = link[2][6].exist = link[3][7].exist =
        link[3][8].exist = link[4][9].exist = link[4][10].exist =
        //draw 3rd stage links:
        link[5][11].exist = link[5][12].exist = link[6][13].exist =
        link[6][14].exist = link[7][15].exist = link[7][16].exist =
        link[8][17].exist = link[8][18].exist = link[9][19].exist =
        link[9][20].exist = link[10][21].exist= link[10][22].exist = true;

        for (int i=1; i<nodecount; i++)
                for (int j=i+1; j<nodecount; j++)
                {
                        if (link[i][j].exist)
                                link[j][i].exist = true;
                        if (link[j][i].exist)
                                link[i][j].exist = true;
                }
        repaint();
}

        /*simulates the bandwidth QoS algorithm with random destinations*/
        public void simulateBwQoS()
        {
                for(int l=0;l<30;l++)
                {
                        dest = (int)(Math.random()*(nodecount-3)+2);
                        runBwQoS();
                        repaint();
                        for (int r=1; r<100000000; r++)//for delay
                        {
                        }
                }
                parent.display.area1.setText("DONE!");
        }

        /*simulates the Dijkstra's algorithm with random destinations*/
        public void simulateDijkstra()
        {
                for(int l=0;l<30;l++)
                {
                        dest = (int)(Math.random()*(nodecount-3)+2);
                        runDijkstra();
                        repaint();
                        for (int r=1; r<100000000; r++)     //for delay
                        {
                        }
                }
                parent.display.area1.setText("DONE!");
        }

}//end PlayGround
```

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
/*a bar with buttons and combo boxes of different commands*/
class MenuBar extends Panel implements ActionListener
{
        // set of options at the bottom of the screen
   JComboBox b1 = new JComboBox();
        JComboBox b2 = new JComboBox();
        Button b3 = new Button("Clear");
   Button b4 = new Button("Lock/Unlock");
        Button b5 = new Button("Reset");
        Button b7 = new Button("Examples");
   JComboBox b6 = new JComboBox();

   QoS parent;
        boolean runQoS = false,    runDij = false,    ex1 = false;

        /*constructor*/
        MenuBar(QoS myparent)
        {
                parent = myparent;
                //add information options:
                b6.addItem("INFORMATION");    b6.addItem("Draw Nodes");
          b6.addItem("Draw Links");       b6.addItem("Change Bandwidth");
          b6.addItem("Delete Nodes");     b6.addItem("Delete Links");
          b6.addItem("Move Nodes");       b6.addItem("Source Node");
          b6.addItem("Destination Node"); b6.addItem("Bandwidth QoS");
          b6.addItem("Established Flows");          b6.addItem("Blocked Flows");
                //add run options:
                b1.addItem("Run");        b1.addItem("Dijkstra");     b1.addItem("Bw QoS");
                //add simulation options:
                b2.addItem("Simulate");   b2.addItem("Dijkstra");
                b2.addItem("Bw QoS");
                setLayout(new GridLayout(1, 7, 10, 0));
                b1.addActionListener(this);                    add(b1);
                b2.addActionListener(this);                    add(b2);
                b3.addActionListener(this);                    add(b3);
                b4.addActionListener(this);                    add(b4);
                b5.addActionListener(this);                    add(b5);
                b7.addActionListener(this);                    add(b7);
                b6.addActionListener(this);                    add(b6);
        }
        public void actionPerformed (ActionEvent evt)
        {
                if (evt.getSource() instanceof Button)
                {
                        //if (evt.getActionCommand().equals("Clear"))
                        if (!parent.playground.Locked&&(evt.getSource() == b3))
                        {
                                parent.playground.clear(); runQoS = runDij = false;
                                parent.playground.runQoS = parent.playground.runDij = false;
                                parent.playground.set = false;        parent.playground.dest = 0;
                                parent.display.area1.setText("");
                                parent.playground.flows = parent.playground.blocks = "";
                                parent.playground.flowNum = parent.playground.blockCount = 0;
                                parent.playground.blockingAt = "";
                                parent.playground.req = 0; b6.setSelectedIndex(0);
                        }
```

```
if (evt.getSource() == b4)
        {
                if (parent.playground.Locked)
                {
                        parent.playground.unlock();
                        parent.playground.runQoS = parent.playground.runDij = false;
                        parent.playground.repaint();          b6.setSelectedIndex(0);
                        parent.display.area1.setText("READY");

                }
                else
                {
                        parent.playground.lock();  b6.setSelectedIndex(0);
                        parent.display.area1.setText("The System is Locked!!"+

        "\nPress Unlock or Clear");
                }
        }
        if (!parent.playground.Locked&&(evt.getSource() == b5))
        {
                parent.playground.reset();  runQoS = runDij = false;
                parent display.area1.setText("READY");
                parent.playground.flows = parent.playground.blocks = "";
                parent.playground.flowNum = parent.playground.blockCount = 0;
                parent.playground.blockingAt = "";
                parent.playground.req = 0; b6.setSelectedIndex(0);
        }
        if (!parent.playground.Locked&&(evt.getSource() == b7))
        {
                if (!ex1)
                {
                        parent.playground.example1();       runQoS = runDij = false;
                        parent.display.area1.setText("READY");
                        parent.playground.flows = "";parent.playground.flowNum = 0;
                        b6.setSelectedIndex(0);     ex1 = true;
                }
                else
                {
                        parent.playground.example2();       runQoS = runDij = false;
                        parent.display.area1.setText("READY");
                        parent.playground.flows = "";parent.playground.flowNum = 0;
                        b6.setSelectedIndex(0);     ex1 = false;
                }
        }

}// big if

if (evt.getSource() instanceof JComboBox)
{
        if (b1.getSelectedItem().toString().equals("Bw QoS"))
        {
                if(!parent.playground.Locked)
                {
                 runQoS = true;
                 parent.playground.runBwQoS();    parent.playground.repaint();
                 b1.setSelectedIndex(0);     parent.playground.unlock();

           }
        }
```

```
if (b1.getSelectedItem().toString().equals("Dijkstra"))
{
        if(!parent.playground.Locked)
        {
         runDij = true;
         parent.playground.runDijkstra();   parent.playground.repaint();
         b1.setSelectedIndex(0);   parent.playground.unlock();
  }
}
if (b2.getSelectedItem().toString().equals("Dijkstra"))
{
        if(!parent.playground.Locked)
        {
         parent.playground.simulateDijkstra();        b2.setSelectedIndex(0);
  }
}
if (b2.getSelectedItem().toString().equals("Bw QoS"))
{
        if(!parent.playground.Locked)
        {
         parent.playground.simulateBwQoS();        b2.setSelectedIndex(0);
  }
}
if (b6.getSelectedItem().toString().equals("Draw Nodes"))
{
        parent.display.area1.setText("To Draw Nodes:\n"+
         "Click where you want the node to be");
        b6.setSelectedIndex(0);
}
if (b6.getSelectedItem().toString().equals("Draw Links"))
{
        parent.display.area1.setText("To Draw Links:\n"+
         "Drag mouse from one node to another");
        b6.setSelectedIndex(0);
}
if (b6.getSelectedItem().toString().equals("Change Bandwidth"))
{
        parent.display.area1.setText("To Change Bandwidth:\n"+
        "Type the new bandwidth value on the left text field\n"+
                        "then click the ball on the link");
        b6.setSelectedIndex(0);
}
if (b6.getSelectedItem().toString().equals("Delete Nodes"))
{
        parent.display.area1.setText("To Delete Nodes:\n"+
         "Click the node with mouse button #2");
        b6.setSelectedIndex(0);
}
if (b6.getSelectedItem().toString().equals("Delete Links"))
{
        parent.display.area1.setText("To Delete Links:\n"+
         "Change Link Bandwidth to a -ve number");
        b6.setSelectedIndex(0);
}
if (b6.getSelectedItem().toString().equals("Move Nodes"))
{
        parent.display.area1.setText("To Move Nodes:\n"+
        "SORRY! this will be implemented later");
        b6.setSelectedIndex(0);
}
```

```
                if (b6.getSelectedItem().toString().equals("Source Node"))
                {
                        parent.display.area1.setText("The Source Node:\n"+
                        "This is the blue node where the algorithm is carried out\n"+
                                                        "It can not be deleted!");
                        b6.setSelectedIndex(0);
                }
                if (b6.getSelectedItem().toString().equals("Destination Node"))
                {
                        parent.display.area1.setText("The Source Node:\n"+
                        "This is set by right clicking the node\n"+
                                                        "It is green in colour");
                        b6.setSelectedIndex(0);
                }
                if (b6.getSelectedItem().toString().equals("Bandwidth QoS"))
                {
                        parent.display.area1.setText("To Bandwidth QoS:\n"+
                        "A user MUST specify this to obtain the QoS paths to the"+
                                                         " destinations");
                        b6.setSelectedIndex(0);
                }
                if (b6.getSelectedItem().toString().equals("Established Flows"))
                {
                        parent.display.area1.setText("Established Flows ("+
                                parent.playground.flowNum+"):\n"+parent.playground.flows);
                        b6.setSelectedIndex(0);
                }
                if (b6.getSelectedItem().toString().equals("Blocked Flows"))
                {
                        parent.display.area1.setText("Blocked Flows ("+
                                        parent.playground.blockCount+"):\n"+
                                                parent.playground.blocks + "\n At req: " +
                                                        parent.playground.blockingAt);
                        b6.setSelectedIndex(0);
                }
        }

        }//actionPerformed
}//end class MenuBar
```

```
import java.awt.*;

public class QoS extends java.applet.Applet
{
        PlayGround playground = new PlayGround(this);
        MenuBar menubar = new MenuBar(this);
        Display display = new Display(this);
  public void init()
        {
        setLayout(new BorderLayout(20, 20)); //Constructs a border layout
        //add the components to an applet:
        add("Center", playground);
        add("South", menubar);
        add ("North", display);
        }

  public Insets getInsets()
        {
                setBackground(Color.GRAY);      return new Insets(20,20,20,20);
  }
} //end QoS

import java.awt.*;          import java.awt.event.*;    import javax.swing.*;

class Display extends Panel implements ActionListener
{
        // components at the top of the screen:
        JLabel label1 = new JLabel("Enter Bandwidth Here (then click the link):");
        JLabel label2 = new JLabel("Bandwidth QoS:");
        JTextField text1 = new JTextField(3);       //link bandwidth input area
        JTextField text2 = new JTextField(3);       //QoS bandwidth input area
        TextArea area1 = new TextArea(3,50);        //display window

        QoS parent;
  Display(QoS myparent)
        {
                parent = myparent;

                add(label1);
                text1.addActionListener(this);      add(text1);
                add(area1);         add(label2);
                text2.addActionListener(this);      add(text2);
        }

public void actionPerformed (ActionEvent evt)
{
}
}// Display
```

```
public class Link
{
int bwidth, //available bandwidth
        bw,                 //link bandwidth
        ballX,
        ballY,
        weight;  //weight of the link
boolean exist;

public Link ()
{
        ballX = ballY = -30;
        exist = false;
        bwidth = bw = 10;
        weight = 1;         //since weight is this program is just a hop-count

}
}//end Link

public class Node
{
int xPos , yPos ;
char name;

public Node()
{
    xPos = -30;     //outside the
   yPos = -30;      //playground
   name = '?';
}
public Node(int x, int y)
{
      xPos = x;       yPos = y;       name = '?';
}

}//end Node

public class tab_entry
{
        int bw;
        Object neighbor;
        String path;

        public tab_entry ()
        {
                neighbor = null;
                bw = 0;  path = "";
        }

}//end tab_entry
```