

1205

**A COMPARATIVE STUDY OF MODIFIED AND
EXISTING ARPANET
ROUTING ALGORITHM USING SIMULATION**

Dissertation submitted to

*Jawaharlal Nehru University
in partial fulfillment of the requirements
for the award of the Degree of*

MASTER OF TECHNOLOGY

IN

COMPUTER SCIENCE AND TECHNOLOGY

DHARMESH KR. SRIVASTAVA


**SCHOOL OF COMPUTER AND SYSTEM SCIENCES
JAWAHARLAL NEHRU UNIVERSITY
NEW DELHI - 110 067**

JANUARY, 1995

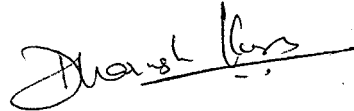
CERTIFICATE

This is to certify that the thesis entitled "A COMPARATIVE STUDY OF MODIFIED AND EXISTING ARPANET ROUTING ALGORITHM USING SIMULATION" being submitted by me to Jawaharlal Nehru University, in partial fulfillment of the requirements for the award of the degree of Master of Technology in Computer Science and Technology is a record of original work done by me under the supervision of Prof. Karmeshu & Dr. Madan, School of Computer and system sciences during the Monsoon Semester, 1994.

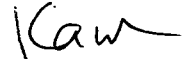
The result reported in this thesis have not been submitted in part or full to any other University or Institution for the award of any degree etc.


16-1-95

Prof. K.K. Bharadwaj
Dean, SCSS,
J.N.U, New Delhi



(DHARMESH KR. SRIVASTAVA)



Prof. Karmeshu
Professor, SCSS
J.N.U., New Delhi



Prof. S. Madan
Asst. Professor,
SCSS, JNU
New Delhi

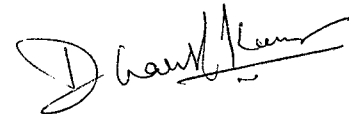
ACKNOWLEDGEMENT

I express my deep sense of gratitude towards my project guide **Dr. Karmeshu & Dr. S. Madan** of Computer Science Department, J.N.U Delhi under whose invaluable guidance and incessant encouragement my work has taken its present shape. I gratefully acknowledge the facilities provided by the Computer Science Department.

I am indebted to **Dr. S. Madan**, under whose dynamic guidance, cooperation and constant encouragement this project has seen the light of the day.

My thanks are also due to those who are not mentioned but have been associated with this work.

Date : 11th January, 95



(Dharmesh Kr. Srivastava)

ABSTRACT

In this desertation the routing algorithm of ARPANET has been studied and a modified algorithm suggested to improve its performance measures of throughput and delay.

In the presently running algorithm when the cost of a link changes significantly several routes passing through it are re-routed. It is felt that by doing so several of the re-routes may pass through the same link (which previously had less traffic), raising its traffic significantly. This may cause deterioration in throughput and delay. This has been overcome in the modified routing algorithm suggested by rerouting only one path at a time.

The present algorithm uses delay over links as a cost metric to determine the shortest paths for the purpose of routing. Such a metric has been found to be inadequate as the increase in delay is disproportionate in relation to increase in resource utilization causing uneven distribution of traffic leading to deterioration in performance. An alternative cost metric using the link utilizations and the number of hops in the path has been suggested.

Since performance of an adaptive routing algorithm (as in ARPANET) is difficult to determine analytically, the performance evaluation is carried out by simulation models any by comparing with the results of the simulation model of the existing routing algorithm in ARPANET.

Various combination of the routing algorithm and cost functions were simulated and their performance was examined. The model with the cost function taken as the product of maximum utilization of a link in the path and number of hops in the path with the routing algorithm taken as the single re-route algorithm suggested, was found to give the best results.

CONTENTS

1.	BASICS OF COMPUTER NETWORKS	1
1.1	WHY ARE NETWORKS REQUIRED	1
1.2	COMPUTER NETWORK CLASSIFICATION SCHEMES	2
1.3	THE ISO OSI SEVEN LAYER MODEL	5
1.4	CIRCUIT SWITCHING AND PACKET SWITCHING	7
1.5	ROUTING IN PACKET SWITCHED NETWORKS	8
1.6	FUNCTIONS OF ROUTING PROCEDURES	9
1.7	CLASSIFICATION OF ROUTING ALGORITHMS	11
2.	ARPANET ROUTING ALGORITHM	15
2.1	ORIGINAL ROUTING ALGORITHM	15
2.2	NEW ROUTING ALGORITHM	15
2.3	SHORTEST PATH FIRST ALGORITHM	19
2.4	ADDITION TO BASIC ALGORITHM	21
2.5	DELAY MEASUREMENT	22
2.6	UPDATING POLICY	23
2.7	COMPARISON OF OLD AND NEW ALGORITHM	23
3.	SIMULATION OF PROPOSED ARPANET ALGORITHM	25
3.1	INADEQUACIES IN NEW ROUTING ALGORITHM	25
3.2	MODIFIED ARPANET ALGORITHM	25
3.3	SIMULATION MODEL	29
3.4	SIMULATION TECHNIQUES	33

3.5 IMPLEMENTATION DETAILS	36
3.6 SIMULATION RESULTS	37
4. CONCLUSION	54
APPENDIX - I	
BIBLIOGRAPHY	58
APPENDIX - II	
LISTING	61

CHAPTER - 1

BASICS OF COMPUTER NETWORKS

1.1 Why Are Networks Required : Most computer have serial interface referred as "asynchronous interface" or RS-232 port. If terminal emulation software is run on the personal computer, the serial port can be connected to a serial port of another computer. The personal computer can then act as though it is a terminal and operate the other computer.

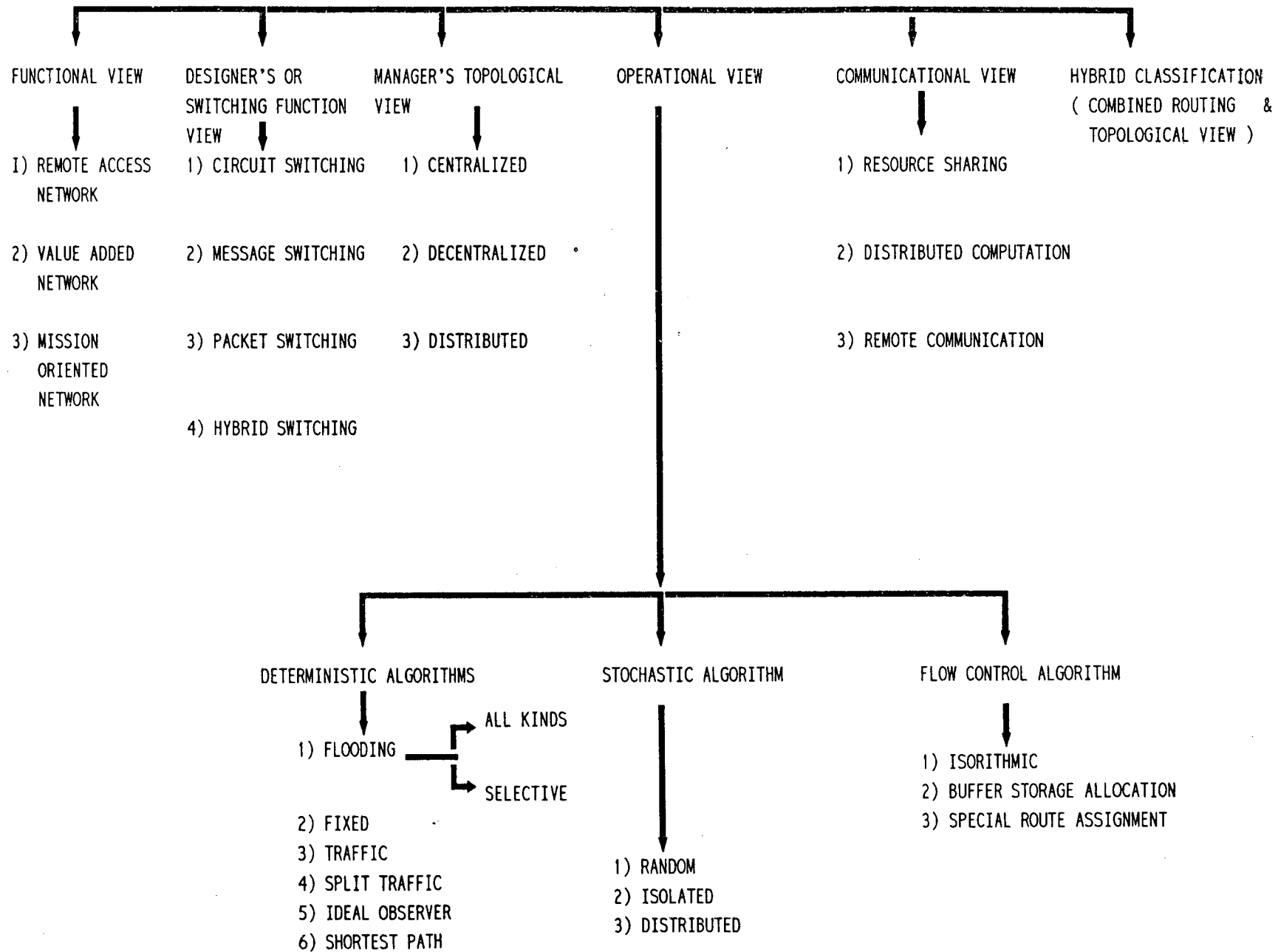
There are problems with this sort of set-up. Firstly, the computer must be close together, as the Rs.232 electrical signalling cannot be driven over very long distances. Secondly, the transfer is limited by the maximum speed at which the serial interface can transfer data. The first problem can be solved with devices called line drivers and modems. These allow much longer links between the computers. Most modems are designed to be connected to the Public switched Telephone Network (PSTN).

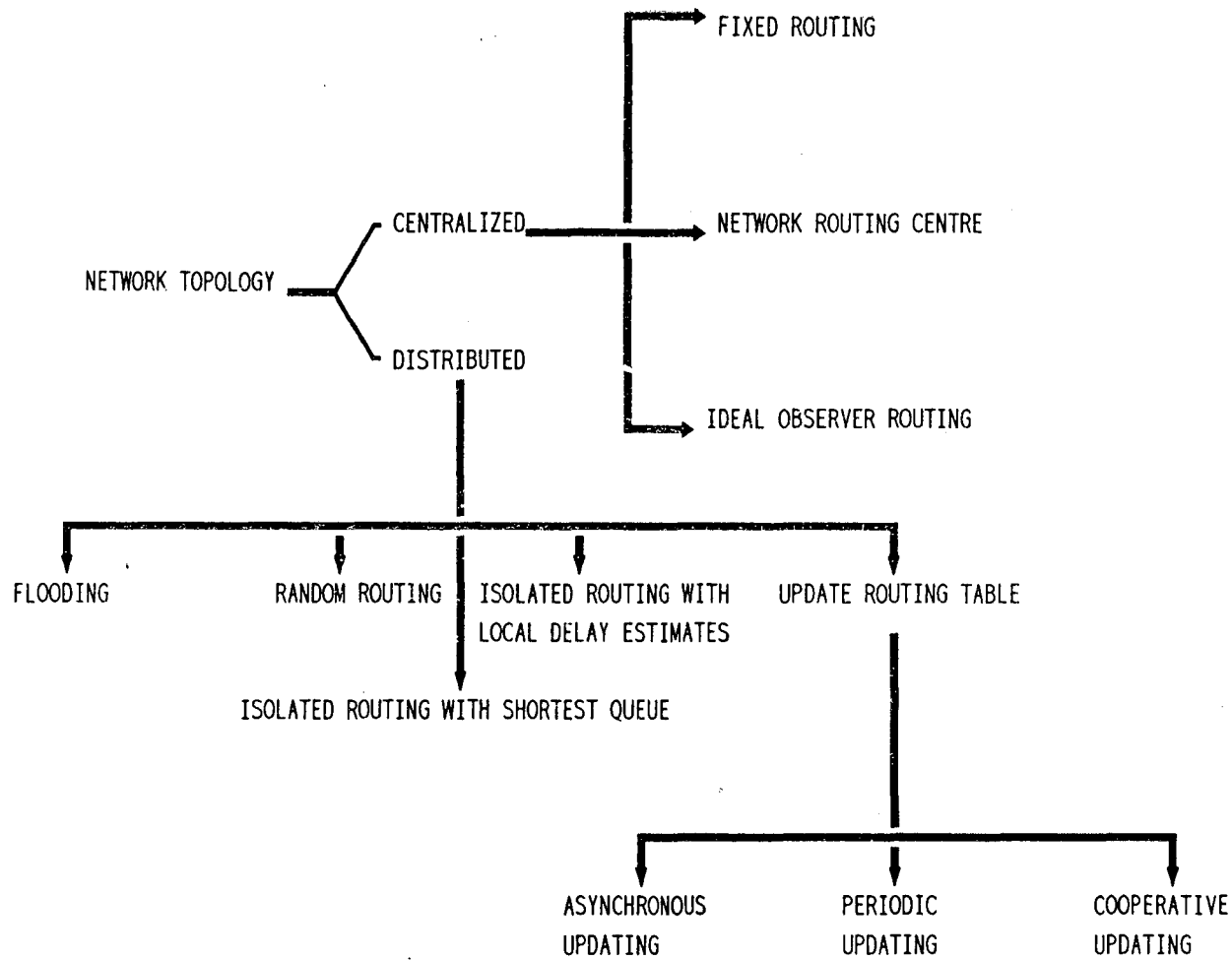
Modems tend to be limited in the rate that they transfer data due to the very low capacity of telephone lines [11].

The solution is to use a network. All of the computers that need to communicate with each other are connected into the network. A link to the remote site is also connected to the network while another network at the remote site connects all the computers there to the remote link. A Computer Network is a set of one or more computers, communication links and terminals interconnected to provide service to a set of users. A node is

a computer system that is attached to, or part of a computer network. Transmission links connect the hosts, nodes, and terminals together to form a network. A path is a series of end-to-end links that establishes a route across a part of the network. The links and nodes along with the essential control software make up the communication subnet, or data network. [4]

1.2 COMPUTER NETWORK CLASSIFICATION SCHEMES : [4]





1.3 The ISO OSI Seven Layer Model

The task of implementing communication systems to cope with the problems encountered in the real world is very large. In order to make things manageable, the task has to be broken down into number of sub-tasks. One such subdivision is the ISO seven layer model. [11]

Layer 7 : Application Layer
Layer 6 : Presentation Layer
Layer 5 : Session Layer
Layer 4 : Transport Layer
Layer 3 : Network Layer
Layer 2 : Link Layer
Layer 1 : Physical Layer

ISO OSI SEVEN LAYER MODEL

1.3.1 LAYER 1 - THE PHYSICAL LAYER :

The physical layer deals with the most fundamental aspect of network connection, i.e. connector types, connector pin-outs electrical signalling and signalling conventions.

1.3.2 LAYER 2 - THE LINK LAYER :

The link layer is responsible for transporting information from the higher layer protocols across the physical layer interface between two devices connected together. The link layer provides mechanisms for detecting errors in the information transferred.

1.3.3 LAYER 3 - THE NETWORK LAYER :

LAYER 3 deals with 'end-point devices' connected to each other across a network that may incorporate many device-device links. It is used to establish connections between end-point devices across a network or interconnected networks [11].

1.3.4 LAYER 4 - THE TRANSPORT LAYER :

The transport layer provides a standard interface between the higher level protocols and the underlying network. It is needed because different network and link layers may have different characteristics.

1.3.5 LAYER 5 - THE SESSION LAYER :

The session layer is the lowest layer that deals with the abstract view of the underlying network. It provides support for the information transfer between applications. This includes a mechanism for restarting information transfer at the correct point after a failure that cannot be transparently corrected at the lower levels.

1.3.6 LAYER 6 - THE PRESENTATION LAYER :

The presentation layer deals with the way in which information being transferred is represented. It incorporates a 'transfer syntax' to define rules for how the information is to be represented so that OSI application can understand the information. [11]

1.3.7 LAYER 7 - THE APPLICATION LAYER :

This layer provides the ultimate in high level support for the application using network. It includes facilities like file transfer, job transfer and message handling.

1.4 CIRCUIT SWITCHING AND PACKET SWITCHING :

There are two fundamentally and competing approaches to communications ; pre-allocation and dynamic allocation of transmission bandwidth. The former is called 'circuit switching' and latter 'packet switching'. [7]

Circuit switching is analogous to the telephone (Voice) network call and message routing are set up prior to commencement of message transmission. Once a complete circuit or route is established the message is ready for transmission. It can be either manual or automatic. In the manual mode, the user dials a sequence of digits to obtain access to a particular computer system. In the automatic switching mode, the required path is established on the basis of information in the data stream.

In message switching a message works its way through the network from link to link, queuing at specific nodal points. Small computers located at message concentration centres and routing points perform the necessary message coding for entry into the network and combine buffer and route messages to the next destination. The networks introduce buffering or queuing delay and thus, delay time or response time plays a critical role in their design. The main advantage of message switching over circuit switching is increased circuit utilization.

Packet switching attempts to solve the problems of variable length messages in message switching by dividing a 'message' into 'packets'. A packet is a subdivision of a message prefaced with an identifier containing suitable address information and information which will the message to be reconstructed. [4]

To use CS or PS depends on a number of factors including mean message size, intermessage arrival, and conversation length.

Hybrid switching is a switching approach which combines CS and PS within a single data network and thus can handle each category of traffic. Networks, providing hybrid switching capabilities apply dynamic time-division multiplexing techniques to allocate a portion of channel bandwidth to CS applications. The remaining channel bandwidth is then available for PS traffic.[4]

1.5 ROUTING IN PACKET SWITCHED NETWORKS :

Routing implies finding the optional path from each node treated as source node to all other nodes treated as destination nodes. The two main functions performed by a routing algorithm are the selection of routes for various origin-destination pairs and the delivery of messages to their correct destination once the routes are selected. [11]

The ideal routing algorithm should use little CPU time and network bandwidth. It must base its decisions on the current state of the network and not on the state of the network some time back. Also it must minimize its use of network bandwidth

to keep its data upto date. Routing determines the response time seen by the user, and in part the efficiency of node and link utilizations.

Routing interacts with flow control in determining throughput (quantity of service) and average packet delay (quality of service) measures, by means of feedback mechanism.

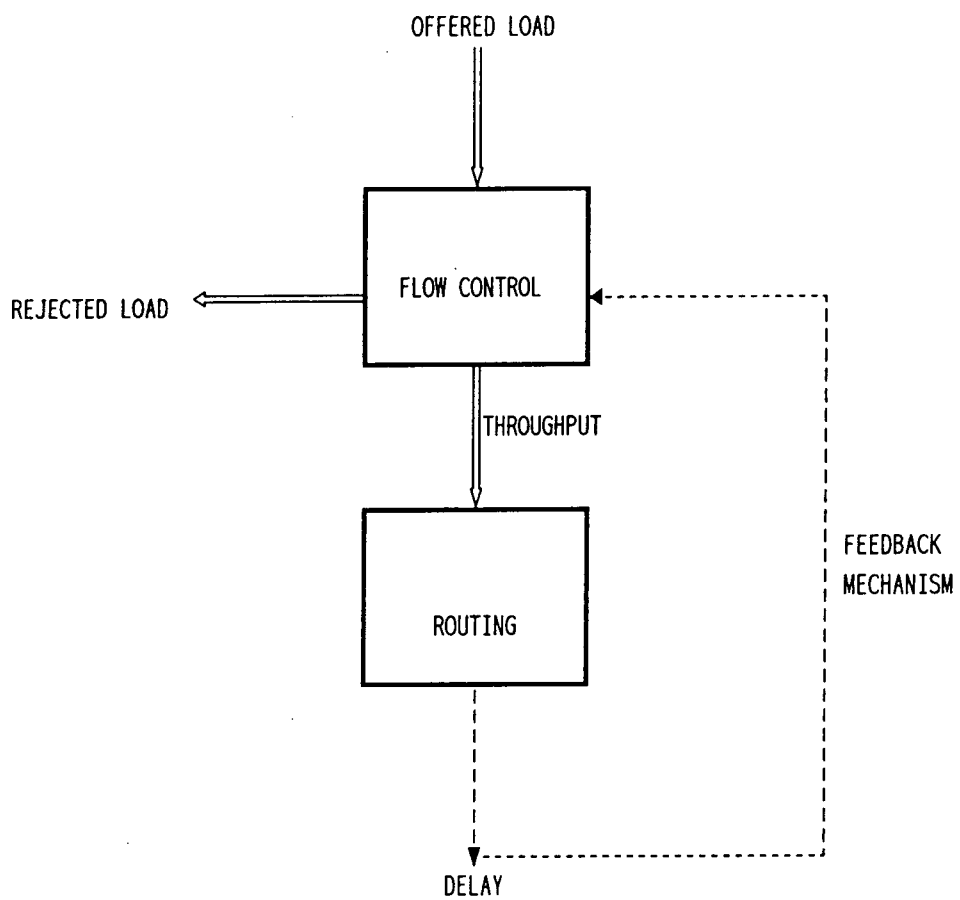
When the traffic load offered by the external sites to the subnet is relatively low, it will be fully accepted into the network. When the offered load is excessive, a portion will be rejected by the flow control algorithm. As the routing algorithm is more successful in keeping delay low, the flow control algorithm allows more traffic into the network. [1]

The effect of good routing is to increase throughput for the same value of average delay per packet under high offered load conditions and to decrease average delay per packet under low and moderate offered load conditions.

1.6 FUNCTIONS OF ROUTING PROCEDURES :

Any adaptive routing procedure must perform a number of functions :

- i) Measurement of the network parameters pertinent to the routing strategy.
- ii) Forwarding of the measured information to the points [Network Control Centre (NCC) or nodes] at which routing computation takes place.



INTERACTION OF ROUTING AND FLOW CONTROL

- iii) Computation of routing tables.
- iv) Conversion of routing table information to packet routing decisions.

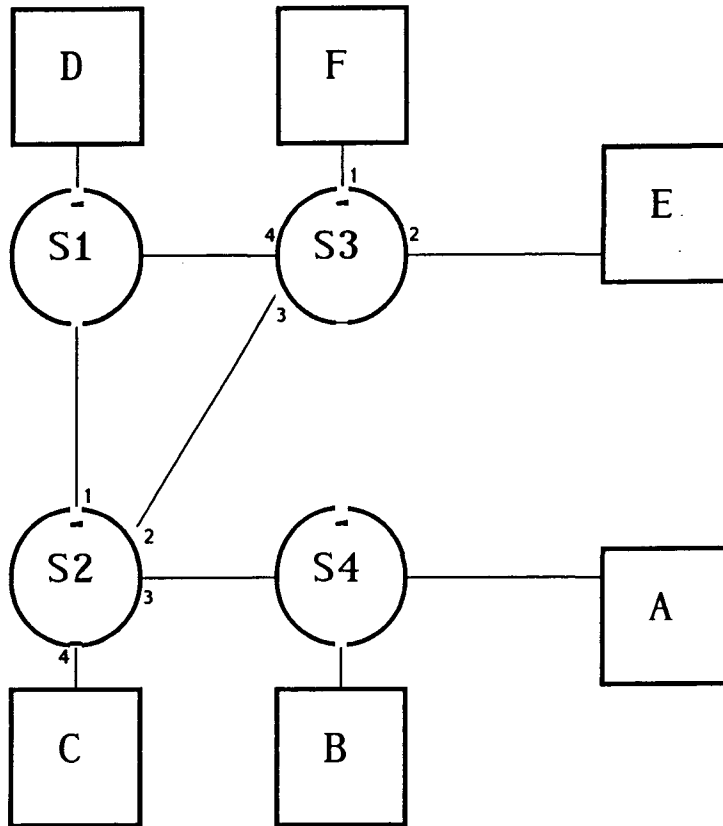
Typical information that is measured and used in routing computation consists of states of communication lines, estimated traffic link delays & available resources (line capacity, nodal buffers, etc.). The pertinent information is forwarded to the NCC in a centralized system and to the various nodes in a distributed system. [5]

1.7 CLASSIFICATION OF ROUTING ALGORITHMS :

There are number of ways to classify routing algorithms. One way is to divide them into centralized and distributed. In centralized algorithms, all route choices are made at a central node, while in distributed algorithms, the computation of routes is shared among the network nodes with information exchanged between them as necessary.

Another classification of routing algorithms relate to whether they change routes in response to the traffic-input patterns.

- (i) Static Routing : In static routing algorithms, the path used by the sessions of each origin-destination pair is fixed regardless of traffic conditions. It can only change in response to a link or node failure.



NETWORK TOPOLOGY

(ii) Adaptive Routing : In adaptive routing the paths used to route new traffic between origins and destinations change occasionally in response to congestion. [1]

There are fundamentally two different types of routing strategy:

(i) Fixed Routing : Fixed routing is the simplest routing strategy in that the packet switches forming the network contain fixed routing tables. These routing tables provide them with all the information that they need to be able to route packets around the network.

The routing table consists of a number of entries one for each end-point device connected to the network. In the above network, there are six end-point devices, so the routing table will have six entries. Each entry contains two pieces of information. The first is the address of the endpoint device, the second is the link down which packets for that end-point device should be routed.

S2's ROUTING TABLE	
Address of end-point device	Link
A	3
B	3
C	4
D	1
E	2
F	2

The disadvantage in this process of building routing tables is that it requires quite detailed knowledge of the topology of the network. [11]

- (ii) Dynamic Routing : In dynamic routing, the packet switches are able to make routing decisions based on conditions in the network when packets are switched.

Advantages of dynamic routing over fixed routing :

- (a) In many network, availability is important. It is important that paths between end point devices should be available as much of the time as possible. To increase availability, the network should be able to automatically use alternate routes between end-point devices in case of link or packet switch failure.
- (b) Another advantages is 'load sharing'. A dynamic routing strategy may well be able to adjust the loading of each route to ensure that maximum use is made of the links and that delay to packets is kept as low as possible.

Distributed Adaptive Routing gets full benefit of dynamic routing, in the sense that the packet switches in the network have information from other parts of the network. The packet switches exchange information about the state of the network, modifying their routing strategies as appropriate.

CHAPTER - 2

ARPANET ROUTING ALGORITHM

ARPANET, a computer network is the acronym for ADVANCED RESEARCH PROJECT AGENCY'S NETWORK. It was created in 1969 under U.S. Department of defence. Beginning with four nodes, it now runs as an operational system with over 100 computers connected to 56 nodes throughout the continental United States, Hawaii and Europe. It is a distributed network with at least two paths between any pair of nodes. Most of its lines are 50-K bit/s synchronous links. It is a store-and-forward packet switched network in which the transport protocol is message oriented. Message longer than the maximum packet length are segmented into up to 8 packets at the source node and are reassembled at the destination node. [5]

The original ARPANET routing algorithm suffered with various problem and a new routing algorithm was implemented later which is being used till date.

2.1 ORIGINAL ROUTING ALGORITHM :

Each packet is directed towards its destination along a path for which the total estimated transit time is smallest. This path is not determined in advance. Each interface message processor (IMP) individually decides which line to use in transmitting a packet addressed to another destination. This selection is made by a simple table look-up procedure. For each possible destination, an entry in the routing directory at each IMP designates the appropriate next line in the path.

Each IMP also maintains a network delay table which gives an estimate of the delay it expects a packet to encounter in reaching every possible destination over each of its output lines.

Periodically, every $2/3$ of a second, the IMP selects the minimum delay to each destination and puts it in a minimum delay table. Since all of the neighbours of an IMP are also sending out their minimum delay tables every $2/3$ second, an IMP receives a minimum delay table from each of its neighbors every $2/3$ second. As each table arrives it is read in over the row of the delay table corresponding to the line it arrived on. After all the neighbour's estimates have arrived, the IMP adds its own contribution to the total delay to each destination, to each column of delay table. Thus the IMP has an estimate of the total delay of each destination over the best path. [9]

Advantages of original routing Algorithm :

- (i) The strongest point is that it is simple. The IMP does not have to know the topology of the network. When IMPs and lines go down, the algorithm functions as usual, and the new routing information propagates through the network by a process of exchange between neighbours.
- (ii) It is not costly one in terms of network resources. The program in the IMP picks the minimum delay and hop counts from the routing messages received from each neighbour for all destinations. Thus, the calculation

is proportional to the number of IMP's in the network and number of lines connected to each IMP. The routing computation takes 5% of CPU bandwidth of the IMP.

Disadvantages of Original routing Algorithm :

- (i) The strong bias towards the shortest path is good idea, but the bias makes the algorithm somewhat insensitive to changes in traffic patterns, so that global optimization of delay and throughput is not likely as network load increases.

- (ii) A second fault is that the algorithm maintains only one route per destination, updated every 2/3 second. This means little traffic bifurcation is possible.

Some of the other serious problem encountered are :

- (i) The routing tables were long and increased with the size of the network, leading to an increase in the size of the packets. Transmission of these packets would take considerable time adversely affecting the flow of network traffic.

- (ii) The rate of exchange of routing tables and the distributed nature of calculations causes a dilemma, the network is too slow in adapting to congestion and to important topology changes because of the time taken in such information reaching all nodes, yet it can respond too quickly to minor changes.

The delay measurement process of the old algorithm involves counting the number of packets queued for transmission on its lines and adds a constant to these counts, the resulting number is the 'length' of the line for purpose of routing.

Drawbacks of such scheme :

- i) Suppose two lines have different speeds or different propagation delays, yet the scheme gives the same time delay for both if their queue length are equal.
- ii) There are number of resources e.g. CPU, output line buffer for which a packet may have to wait for significant amount of time before it is queued. Thus queue length is a poor indicator of delay.
- iii) An instantaneous measurement of queue length does not accurately predict average delay because there is significant real time fluctuation in queue length at any traffic level.

The shortcomings have been overcome in the new routing Algorithm. [10]

2.2 NEW ROUTING ALGORITHM :

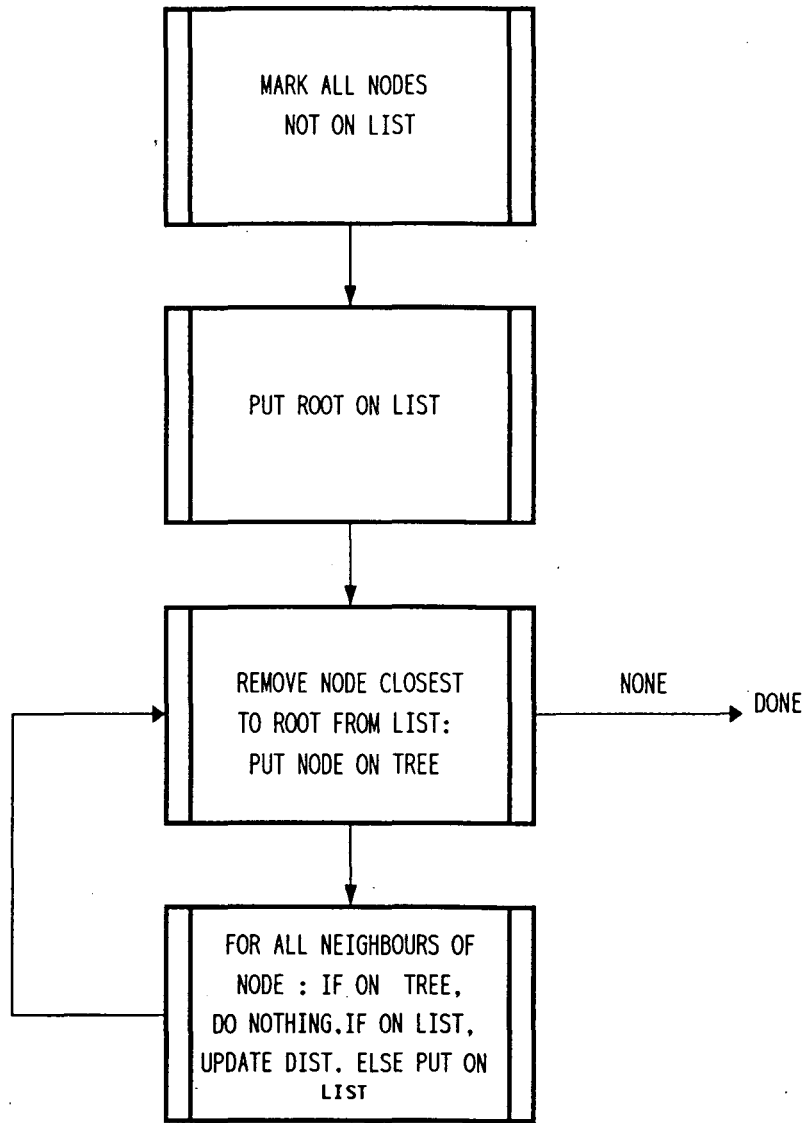
Each node in the network maintains a database describing the network topology and the line delays. Using this database, each node independently calculates the best path to all other nodes, routing outgoing packets accordingly. Because the traffic

in the network can be quite variable, each node periodically measures the delays along its outgoing lines and forwards this information to all other nodes. A routing update generated by a particular node contains information only about the delay on the lines emanating from that node. Hence an update packet is quite small (176 bits on an average) and its size is independent of the number of nodes in the network. An update generated by a particular node travels unchanged to all nodes in the network. Thus updates are small and since they are handled with highest priority, they propagate very quickly through the network, so that all nodes can update their databases rapidly and continue to route traffic in consistent and efficient manner.

2.3 SHORTEST PATH FIRST ALGORITHM :

Routing algorithm used for finding shortest path in the network is SPF (Shortest Path First) algorithm. The basic SPF algorithm uses a database describing the network to generate a tree representing the minimum delay paths from a given root node to every other node of the network. The figure below shows a flowchart of the algorithm.

The database specifies which nodes are directly connected to which other nodes and what the average delay per packet is on each network line. The tree initially consists of just the root node. The tree is then augmented to contain the node that is closest (in delay) to the root and that is adjacent to a node already on the tree. The process continues by repetition. Eventually, the furthest node from the root is added to the tree, and the algorithm terminates. [10]



SHORTEST PATH FIRST ALGORITHM

2.4 ADDITION TO BASIC ALGORITHM :

To handle various possible changes in network status without having to recalculate the whole tree, certain additions are done. For each change, we assume that the shortest path tree rooted at node 'I' prior to the change is known.

- (i) Considering the case where the delay of the line AB from node A to node B increases. If the line is not in the tree, nothing need be done. If the line is in the tree, the delay to B increases, as does the delay to each node whose route from 'I' passes through B.

The first two steps for handling an increase of X in the delay from A to B are as follows:

- (a) Identify nodes in B's subtree and increase their delay from 'I' by X.
- (b) For each subtree node S, examine S's neighbours which are not in the subtree to see if there is a shorter path from 'I' to S via those neighbours. If such path is found , put node S in list.



- (ii) Consider the case where the delay on AB decreases by X. If this line is in the tree, then paths to the nodes of the subtree which have B as its root will be unchanged because the subtree nodes were already at minimum delay, and hence the decreased delay will only shorten their distances from 'I'. However nodes which are not in the subtree and which are farther from 'I' may have a shorter distance via one of the subtree nodes.

TH-5604

2.5 DELAY MEASUREMENT :

When packet arrives at an IMP it is time stamped with its arrival time. When the first bit of the packet is transmitted to the next IMP the packet is stamped with its sent time. Delay time being calculated as

$$\text{Delay Time} = (\text{Sent Time} - \text{Arrival Time} + \text{Propagation Delay} + \text{Transmission Delay}).$$

Routing update is generated only if the average delay measured is significantly different from the last update that was sent.

Since any significant change should be reported and any small change in delay which lasts for a long time should also be reported a threshold is maintained and if the change is greater than or equal to this threshold an update is generated.

The threshold is initially set to 64 m sec. After each measurement period, the newly measured average delay is compared with the previously reported delay. If the difference does not exceed the threshold, the threshold is decreased by 12.8 msec. Whenever a change in average delay equals or exceeds the threshold an update is generated and the threshold is reset to 64 m sec. Since the threshold will eventually decay to zero an update will always be sent after a minute. [10].

2.6 UPDATING POLICY :

Each node measures the actual delay of each packet flowing over each of its outgoing lines and calculates the average delay every 10 sec. Only if this delay is significantly different from the previous one, it is reported to all other nodes unlike the earlier algorithm which allowed the delay estimates to change every 128 ms. Such a small period is inappropriate for obtaining an accurate estimate of delay.

Each update is transmitted to all nodes by the simple method of transmitting it on all lines adjacent to a node. When a node receives an update, it first checks if it has processed that update before. If not, it is immediately forwarded to all adjacent nodes.

As the updates are short and are generated infrequently this procedure uses little line or node bandwidth. Since all nodes perform the same calculation on an identical database there are no permanent routing loops. Transient loops however formed when a change is being processed. [10]

2.7 COMPARISON OF OLD AND NEW ALGORITHM :

(i) -- The old algorithm was distributed in the sense that the routing computation was performed by every node. The inputs to the computation at one node were the outputs at neighboring nodes. Thus old routing was global with each node performing part of it. The SPF is a local computation. Thus the advantages of distributed routing are kept while dispensing with the disadvantages of distributed computation.

- (ii) An important deficiency of old algorithm was its slow response to topological changes. The old algorithm took many seconds to respond to node or link failure. The new algorithm did not observe any congestion arising due to slow response.
- iii) The update of the old routing algorithms were over 1200 bits long. The new routing update averages 176 bits.
- (iv) The old routing algorithm took a fixed amount of time (15-20 ms) to process an update. The new algorithm takes a variable amount of time with the amount of time proportional to the size of routing change necessitated by the update. This result in a much more efficient use of CPU.
- (v) The old algorithm was prone to form loops. A given packet might be trapped in such a loop for a significant amount of time. While the new algorithm cannot be said to be loop-free the loops that form occur only as transients while the network is adapting to routing changes.
- (vi) The new algorithm does take about three times the memory as the old one, but conservation of memory is not generally considered to be important desideratum. [9,10]

CHAPTER 3

SIMULATION OF PROPOSED ARPANET ALGORITHM

3.1 INADEQUACIES IN NEW ROUTING ALGORITHM :

- (i) Increase in resource utilization results in a disproportionate increase in delay. Consequently the distribution of the traffic through the network becomes uneven which may lead to congestion even if the network load is far below its capacity, as due to uneven distribution the load at some switches may increase disproportionately which may cause the packet arrival rate at a link to be greater than the mean delay time, queues would then grow affecting the throughput and delay.

- (ii) In the new algorithm, when the cost of a link AB increases, all the nodes in the subtree of node B are re-routed. And when the cost of the link decreases all nodes, for which the cost decreases are rerouted. In such a scheme several rerouted routes may pass through the same link, raising the system cost to vary arbitrarily. In trying to reduce the cost of one link the cost of some other link may be raised arbitrarily. Thus the centre of congestion shifts from one area of network to another. Thus moving from one unstable condition to another.

3.2 MODIFIED ARPANET ALGORITHM :

To overcome the above inadequacies, a modified algorithm has been suggested:

- (i) The time delay used as cost metric is an inappropriate measure as it does not proportionately vary with resource utilization causing uneven distribution of traffic.

In the proposed cost function link utilization has been used as a cost metric instead of delay. The cost function proposed by **BACHINI and SHEN** uses maximum cost of the link in the path as cost function. In such a case two paths with different number of links but the same maximum utilization will have the same cost.

To overcome this, the cost function is proposed to be the product of maximum link utilization in a path and the number of hops.

$$C(P) = \max_{l_{ij} \in P} (U_{ij}) \times (\text{no. of hops in } P)$$

where P is a path from source to destination.

U_{ij} denotes utilization of link ij where $l_{ij} \in P$.

If two paths are found to have the same cost then the distinguishing criterion will be the number of hops when maximum utilization is less than 0.8 and utilization alone for maximum utilization greater than 0.8. The value 0.8 is taken as beyond this the system tends to saturate.

- (ii) In new algorithm, when the cost of link AB increases, all the nodes in subtree of node B are rerouted. Similarly when

the cost of link decreases, all the nodes for which the cost decreases are rerouted. Since all the nodes are re-routed simultaneously, the centre of congestion shifts from one area to another.

To overcome the above problem, it is preferable to re-route only one route at a time.

The modified algorithm is as follows:

- (i) When utilization of link AB increases,
 - (a) If the line AB is not in the tree, nothing need to be done.
 - (b) If the line AB is in tree, then the cost of reaching node B and its subtree nodes increase, thus the cost of node B and its subtree nodes is updated.
 - (c) As the cost of only subtree nodes of B has increased only they need be considered for rerouting. The rerouting being considered in the descending order of link utilizations of links leading to the subtree nodes. Once a single reroute is found rerouting is stopped.

- (ii) When utilization of line AB decreases,
 - (a) If this line is not in the tree, compute cost of B via AB and reroute B if the cost computed is less than previous cost.
 - (b) If the line AB is in the tree recompute costs of node B and its subtree nodes as it has decreased.

(c) Since the cost of B and its subtree nodes decreases there is no need to reroute them. Only the non-subtree nodes of B may be rerouted through the subtree nodes of B if the cost of these nodes decreases. The rerouting is stopped once a single reroute is found.

3.3 SIMULATION MODEL

Analysis of adaptive routing algorithms is extremely complex; in a distributed network the study of adaptive routing involves the study of time varying behaviour of a set of interactive queues. Due to the involved complexity, simulation is relied on almost exclusively to investigate various algorithms and to carry out comparative studies of different strategies.

The performance of the proposed algorithm is thus evaluated. Using the proposed cost function, sum of utilizations of links in a path as another cost function and the time delay cost function used in ARPANET as another, in combination with the presently running routing algorithm suggested four simulation models that have been developed.

Model 1 replicates the ARPANET routing algorithm and cost function.

Model 2 uses the proposed cost function and modified routing algorithm.

Model 3 uses the sum of link utilizations as a cost function and the ARPANET routing algorithm.

Model 4 uses the sum of link utilizations as a cost function and the modified routing algorithm.

The performance of these models has been evaluated on the basis of throughput and delay measures obtained from the simulated models.

3.3.1 THE SIMULATION PROBLEM

The simulation model developed should represent a real time network as closely as possible thus the simulation model should randomly generate packets at the various nodes in the network. Generate update packets according to the update procedure followed as explained in chapter II. Route the data and update packets, adaptively compute the routing tables and update the destination tables and it also has to maintain statistical data for performance evaluation.

The input information required for the model is

- i) The Network Topology
- ii) Link capacities in bits/sec
- iii) Average interarrival rate at the nodes (in pkts/sec).
- iv) Update packet size (in bits)
- v) Data packet size (in bits)
- vi) The type of routing algorithm to be used.

vii) The updating interval in secs.

viii) Simulation period in secs.

The Statistical information to be computed is

i) Total percentage of packets delivered.

ii) Total percentage of packets rejected.

iii) Average waiting time per packet in each queue.

iv) Average number of packets carried per link.

v) Average number of hops per packet.

vi) Average time per packet spent in the network.

vii) Throughput of the network

viii) Link utilizations and delay on each link.

3.3.2 CONSTRAINTS OF THE SIMULATION MODEL :

The simulation model can not truly represent a real time system. Features of the real time system which do not have a significant affect on the simulation model may be omitted. For eg. CPU processing delay may be neglected for small line capacities as in this case the mean packet interarrival time is essentially greater than the real processing delays.

The simulation of line faults or malfunctions of other components of the system are only of interest when the reliability, availability or security are investigated.

The simulation carried out is only at the network layer as the objective is to evaluate the performance of the routing algorithm only.

Various assumptions have been made for the simulation model:

- i) All links are fully duplex.
- ii) Each link contains one channel only.
- iii) Parallel links are not permitted
- iv) Each node has buffer with finite storage and queues are formed within the buffer area.
- v) Queues are served on first come first serve basis.
- vi) All data packets are of same length and update packets are also of the same length.
- vii) Poisson arrival of packets is taken at the nodes.
- viii) Destination for each packet is generated from a uniform distribution.

- ix) Highest priority is given to update (routing) packets. Next highest priority is given to packets already in the network and then new packets arriving at a node from the terminals.
- x) If the number of packets become more than the network can handle they are dumped.
- xi) A constraint is provided regarding maximum hops a packet can take in the network to prevent indefinite looping.

3.4 SIMULATION TECHNIQUES :

To obtain the throughput - delay characteristics of the modified algorithm a discrete - event simulation model has been designed and developed. Before description of the discrete event simulation model, explanation of certain keywords used are given below :

- a) State : is the collection of variables necessary to describe a system eg. the number of nodes in the network, the queue lengths at various links etc.
- b) Event : is instantaneous occurrence that may alter the system state eg. arrival or departure of a packet from a node.
- c) Discrete system : is a system where the state changes at only finite number of points in time.

d) Dynamic simulation model : is a representative of a system as it evolves over time.

e) Stochastic simulation model : is a model that contains one or more random variables.

Since, the network has a finite number of nodes, the simulation model is required to keep many random variables and hence is necessarily stochastic. The state of the network changes only at a finite time, and it does so in a dynamic manner when an event occurs, therefore a discrete-event simulation model has been used.

The simulation is controlled by means of a simulation block. There are two basic techniques to update time during simulation.

a) Next event time advance : here the clock is advanced to the time of occurrence of the next event at which the system state changes; and the next time of occurrence of an event resulting from the event being serviced is determined. The process repeats until a prespecified condition is satisfied. Periods of inactivity are thus skipped.

b) Fixed increment time advance : here simulation clock is incremented in finite intervals and any occurrence of an event in the interval is serviced.

For event scanning of a sequence of events, there are three general approaches.

- a) Event Scheduling Approach
- b) Process Interaction Method &
- c) Activity Scanning Approach

Next event time advancing scheme has been used for the simulation model developed as it minimises simulation time by skipping periods of inactivity. In this approach events are scheduled by maintaining a linked list of events to be performed in increasing order of their instants of occurrence. The first event in the list thus always being the next event to be performed and any events arising out of it are scheduled by placing them in the list at an appropriate position.

The above is in brief the discrete event simulation model developed. The details are discussed in section (3.5)

There are two approaches to handle this type of simulation.

- 1) Language based approaches.
- 2) Model based approaches.

The language based approaches (eg. Simula, Simgscript) provide users with specialized programming language constructs to support modelling and simulation. The key advantage of these approaches is their generality of applications.

Model based approaches (eg. queueing network simulators) provide users with extensive collection of tools supporting a particular simulation modeling technique. The key advantage of model based approaches is the efficiency with which they may handle large scale simulations by utilizing model specific techniques. The drawbacks of such a scheme are the narrower scope of application of such models and of requiring users to develop an indepth understanding of the modelling techniques.

M/M/1 queueing system was used as a queueing model for the network nodes.

3.5 IMPLEMENTATION DETAILS :

The network configuration, capacities of the links, packet arrival rates at nodes, packet size and simulation period are read from the keyboard or input file. It is then checked whether the network is connected or not. If the network is not connected the input data is read again.

After checking the connectivity of the network, the shortest path algorithm is used to find all shortest paths between all pairs of nodes in the network.

Initially all nodes are waiting for the packets. First arrival time for each node is generated randomly and the event scheduled in the event list. Events are then fetched from the event list and serviced conforming to next event time advance scheme until simulation time is reached.

If a packet is generated at a node, the destination assigned to packet is also randomly generated. The arrival and departure of packets is carried out by the arrival and departures functions developed.

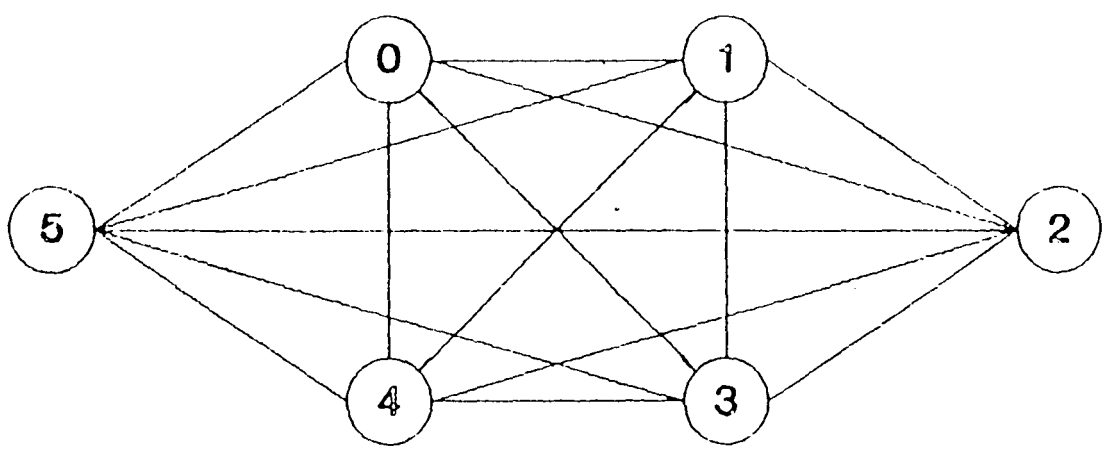
3.6 SIMULATION RESULTS :

The discrete event simulation model technique described in the previous sec 3.4 was used to develop four simulation models described in sec. 3.3.

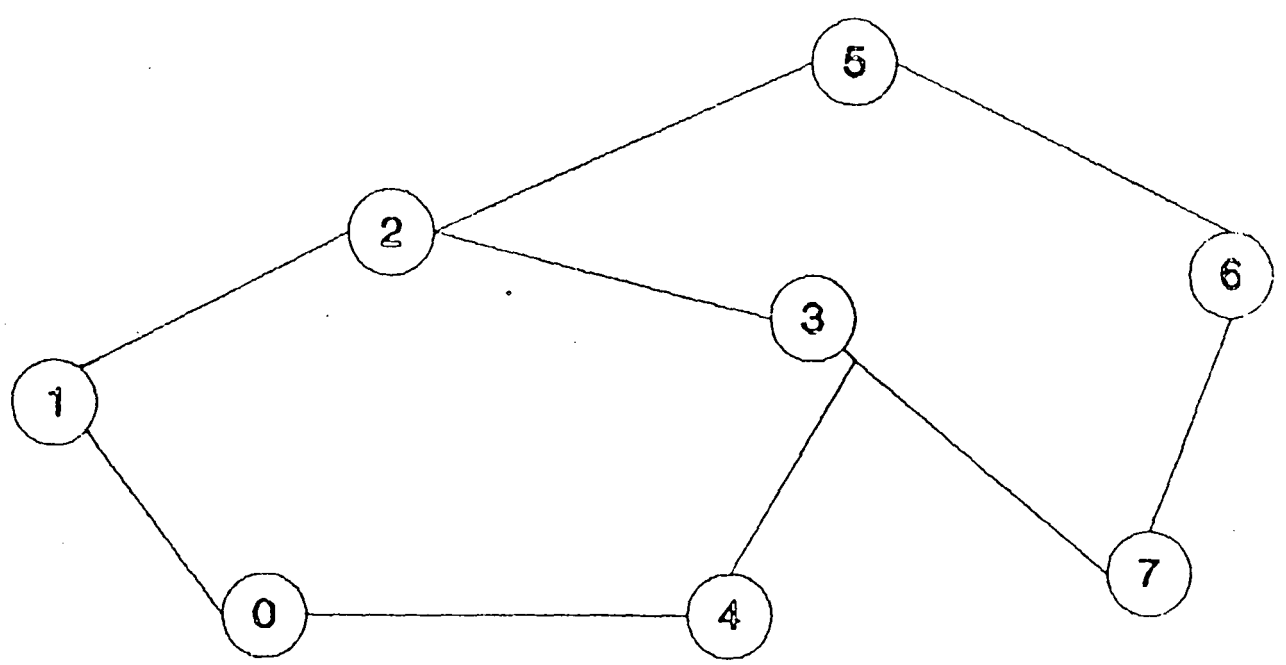
The simulation models gave various statistics like link utilizations, waiting time per queue average waiting time per queue, average waiting time for all queues, mean delay, number of packets generated and that delivered et al. as tabulated in table 3.1

Througput and delay being the major performance measures have been used for evaluation. They have been measured for different arrival rates of packets at nodes.

The throughput and delay for several topologies was examined, the results on an eight node mesh topology and a six node fully connected network topology are given below.



(i) Six node fully connected network



(ii) Eight node mesh network

TOPOLOGIES USED

FIG 3.01

THE NETWORK CONFIGURATION IS

Number of nodes : 6

NETWORK CONNECTIVITY (adjacency matrix)

		DESTINATIONS					
		0	1	2	3	4	5
Source	0 ->		1.00	1.00	1.00	1.00	1.00
Source	1 ->	1.00		1.00	1.00	1.00	1.00
Source	2 ->	1.00	1.00		1.00	1.00	1.00
Source	3 ->	1.00	1.00	1.00		1.00	1.00
Source	4 ->	1.00	1.00	1.00	1.00		1.00
Source	5 ->	1.00	1.00	1.00	1.00	1.00	

LINK CAPACITIES ARE

		DESTINATIONS					
		0	1	2	3	4	5
Source	0 ->		50000.00	50000.00	50000.00	50000.00	50000.00
Source	1 ->	50000.00		50000.00	50000.00	50000.00	50000.00
Source	2 ->	50000.00	50000.00		50000.00	50000.00	50000.00
Source	3 ->	50000.00	50000.00	50000.00		50000.00	50000.00
Source	4 ->	50000.00	50000.00	50000.00	50000.00		50000.00
Source	5 ->	50000.00	50000.00	50000.00	50000.00	50000.00	

Mean interarrival rate (pkt/sec) at nodes are :

0	1	2	3	4	5
100.00	100.00	100.00	100.00	100.00	100.00

Packet Size is : 1008 bits/sec

Network Simulation time is :5 secs

OUTPUT STATISTICS

Total number of message packets generated : 3016

Average number of packets generated : 603.2 packets/sec.

Average number of packets rejected at source: 0.8 packets/sec

Average number of packets rejected in the network: 0.2 packets/sec

Average number of packets delivered (THROUGHPUT) : 599.2 packets/sec

Total % of packets delivered :99.3369

Total % of packets rejected :0.165782

Average number of hops taken per packet:1.0267

Average time spent by each packet delivered in the network : 0.02887

LINK STATISTICS

Average packets delivered by each link in packets/sec

		DESTINATIONS					
		0	1	2	3	4	5
Source	0 ->		36.20	29.40	32.60	28.00	32.00
Source	1 ->	32.60		40.60	32.40	33.80	32.40
Source	2 ->	26.60	35.00		28.20	33.20	33.20
Source	3 ->	33.80	40.00	14.60		36.60	34.60
Source	4 ->	34.00	31.80	32.20	33.40		25.20
Source	5 ->	33.60	32.40	33.00	29.40	31.20	

Average waiting time per packet in QUEUE for each link

		DESTINATIONS					
		0	1	2	3	4	5
Source	0 ->		0.00	0.00	0.00	0.00	0.00
Source	1 ->	0.00		0.00	0.00	0.00	0.00
Source	2 ->	0.00	0.00		0.00	0.00	0.00
Source	3 ->	0.00	0.00	0.00		0.00	0.00
Source	4 ->	0.00	0.00	0.00	0.00		0.00
Source	5 ->	0.00	0.00	0.00	0.00	0.00	

LINK UTILIZATION

		DESTINATIONS					
		0	1	2	3	4	5
Source	0 ->		0.47	0.38	0.43	0.34	0.36
Source	1 ->	0.44		0.62	0.45	0.41	0.42
Source	2 ->	0.34	0.46		0.34	0.42	0.42
Source	3 ->	0.45	0.62	0.20		0.46	0.44
Source	4 ->	0.45	0.38	0.40	0.42		0.33
Source	5 ->	0.45	0.42	0.40	0.36	0.39	

Average waiting time per Queue : 0.415296

COMPREHENSIVE OUTPUT STATISTICS

INPUT PARAMETERS :

Number of nodes : 6
 Simulation Time : 5 secs
 Packet size is : 1008 bits

OUTPUT STATISTICS :

Total no. of message pkts generated : 3016
 Total % of packets delivered : 99.3369
 Total % of packets rejected : 0.165782
 Average waiting time per Queue : 0.415296
 Average number of hops taken / packet : 1.0267
 Average no. of pkts dwd/sec (THROUGHPUT) : 589.2

PLEASE ENTER :

TO UPDATE : 1
 TO RESTART : 2
 TO EXIT ANY KEY EXCEPT 1 & 2
 Enter :

Since models 3 and 4 have the same cost metric and differ in their routing algorithms only, comparison between the performance measures of throughput and delay of the two models reflects on the difference in performance of the two routing algorithms.

Comparisons between the results of models 3 and 4 with 2, reflects on the affect of the cost metric.

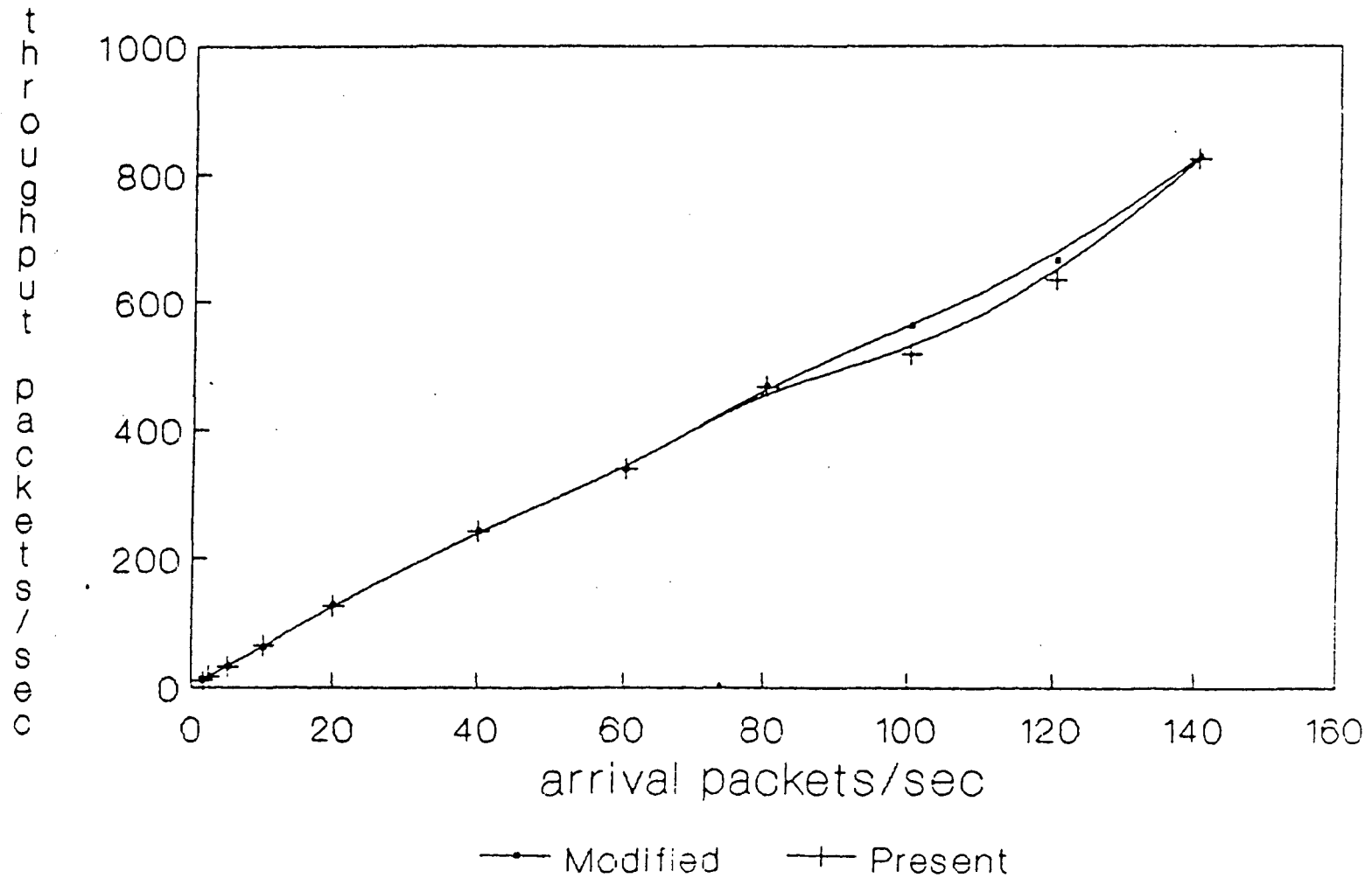
Finally the comparison between models 1 and 2 is done to identify the better performance model.

3.6.1 RESULTS OBTAINED USING A SIX NODE FULLY CONNECTED NETWORK TOPOLOGY.

Throughput and delay comparison graphs with respect to arrival rate have been shown for models 3 and 4 in fig 3.1(a) & (b) respectively. It is observed from fig. 3.1(a) that the throughput increases with increase in arrival rate for both the models, however the throughput shown by model 4 which has the modified algorithm is better than that shown by model 3.

From fig 3.1(b) it is observed that the mean delay increases with increase in arrival rate and then later decreases. The nature of the change in delay with arrival rate shown by both the models is the same: The high delay corresponding to the arrival rate interval of 80 to 120

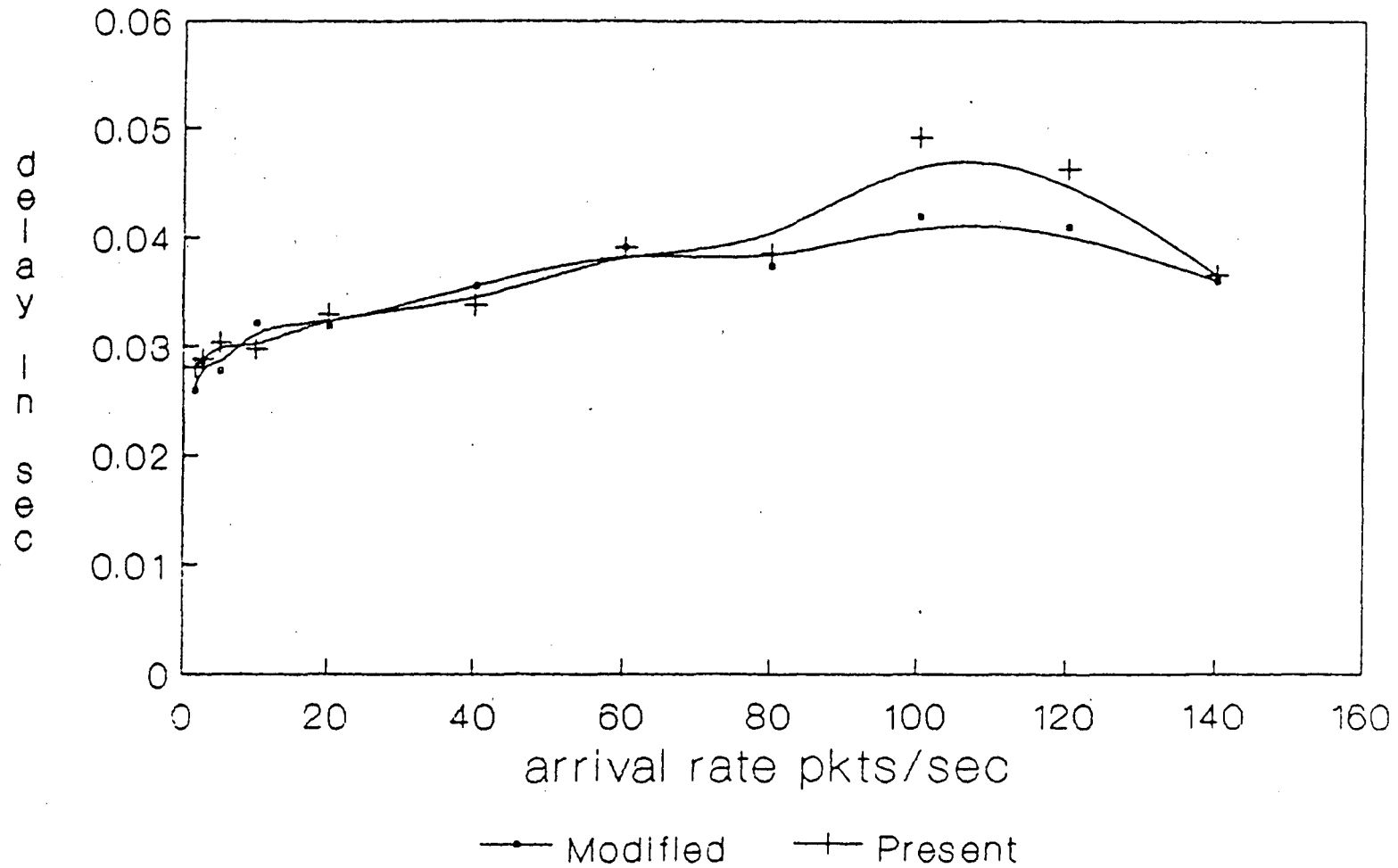
THROUGHPUT COMPARISON



Model 3 Vs Model 4

Fig 3.1 (a)

DELAY COMPARISON



Model 3 Vs Model 4

Fig 3.1 (b)

pkts/sec and the decrease in delay beyond the arrival rate of 100 pkts/sec may be due to the random generation pattern of packets which may have led to high traffic on a particular link used by several routes in that interval. However the delay shown by the modified routing algorithm is found to be less than that of the presently running algorithm on ARPANET.

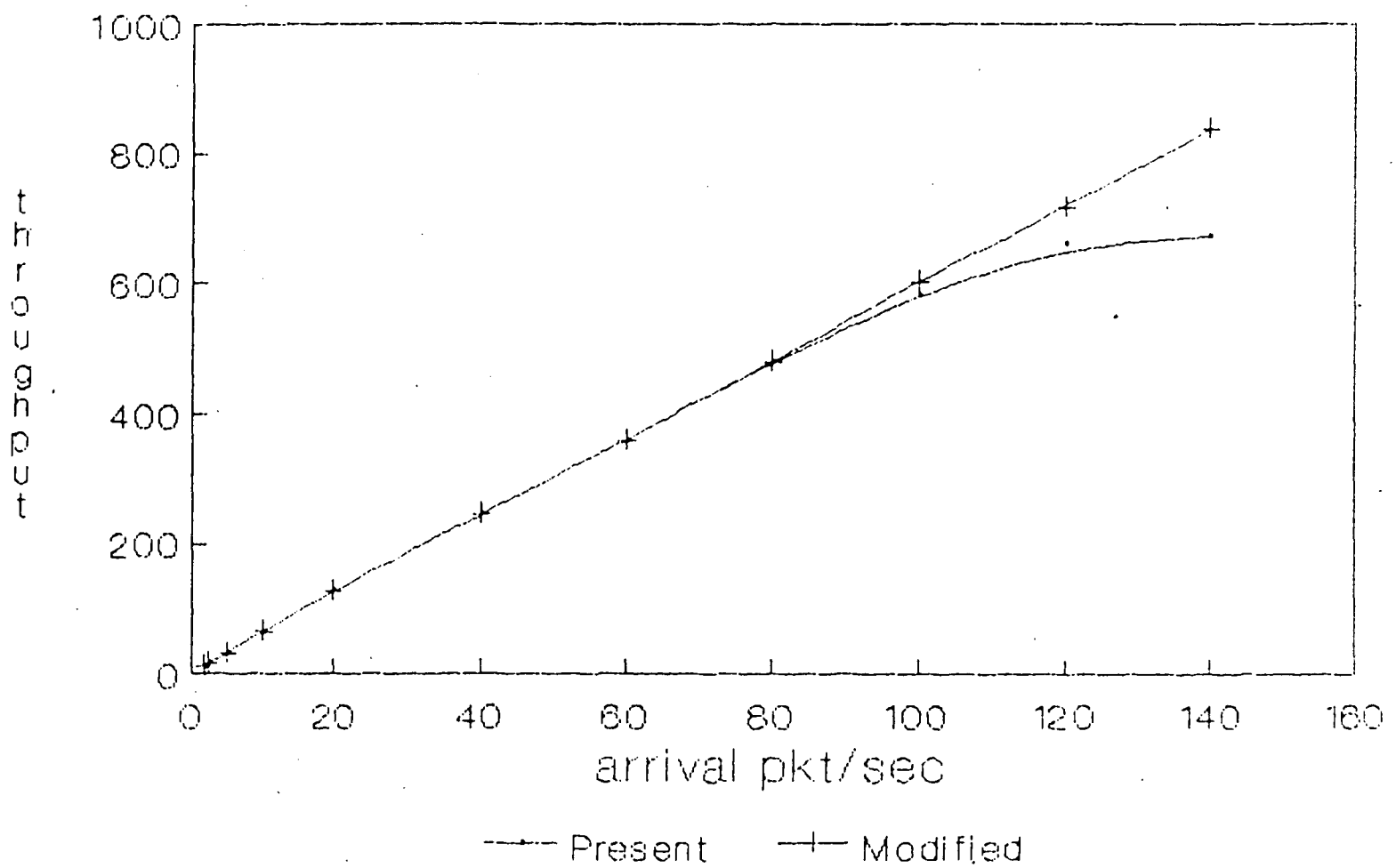
This indicates that the modified routing algorithm gives better performance than the presently running algorithm.

The throughput and delay comparisons of model 1 replicating ARPANET and model 2 having the modified routing algorithm and cost function is shown in figs 3.1(c) & (d) respectively.

The throughput as observed from fig.3.1(c) increases with increase in arrival rate for both the models. However at high traffic beyond 80 packets/sec, the throughput of model 2 is much higher than that of model 1 which approaches congestion as arrival rate is increased to 140 packets/sec.

The delay comparison of the two models shown in fig 3.1(d), shows that the delay increases with increase in arrival rate. The delay being slightly higher for the modified algorithm at low traffic than the presently running algorithm, however at high traffic the delay of the modified

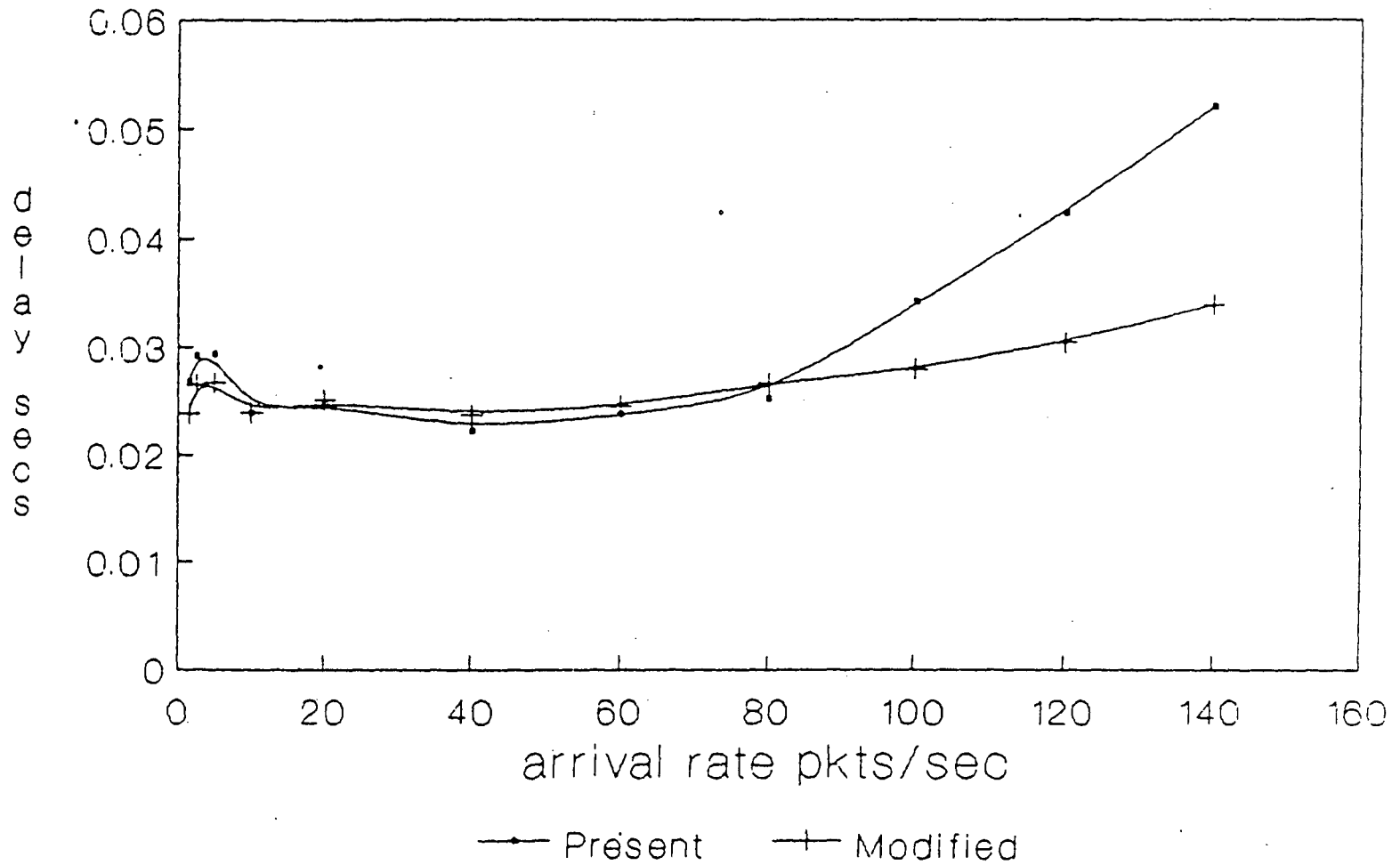
THROUGHPUT COMPARISON



Model 1 Vs Model 2

Fig 3.1 (c)

DELAY COMPARISON



Model 1 Vs Model 2

Fig 3.1 (d)

algorithm is much less than that of presently running algorithm.

The above comparisons indicate that model 2 gives better performance of all the models .

3.6.2 RESULTS OBTAINED USING AN EIGHT NODE MESH NETWORK

The throughput and delay comparison graphs of models 3 and 4 are shown in fig 3.2(a) & (b). The throughput as observed from fig.3.2(a) is found to increase with arrival rate for both the models. At high traffic it is found to be nearly the same for both the models, but at low traffic model 4 shows better performance (higher throughput) than model 3.

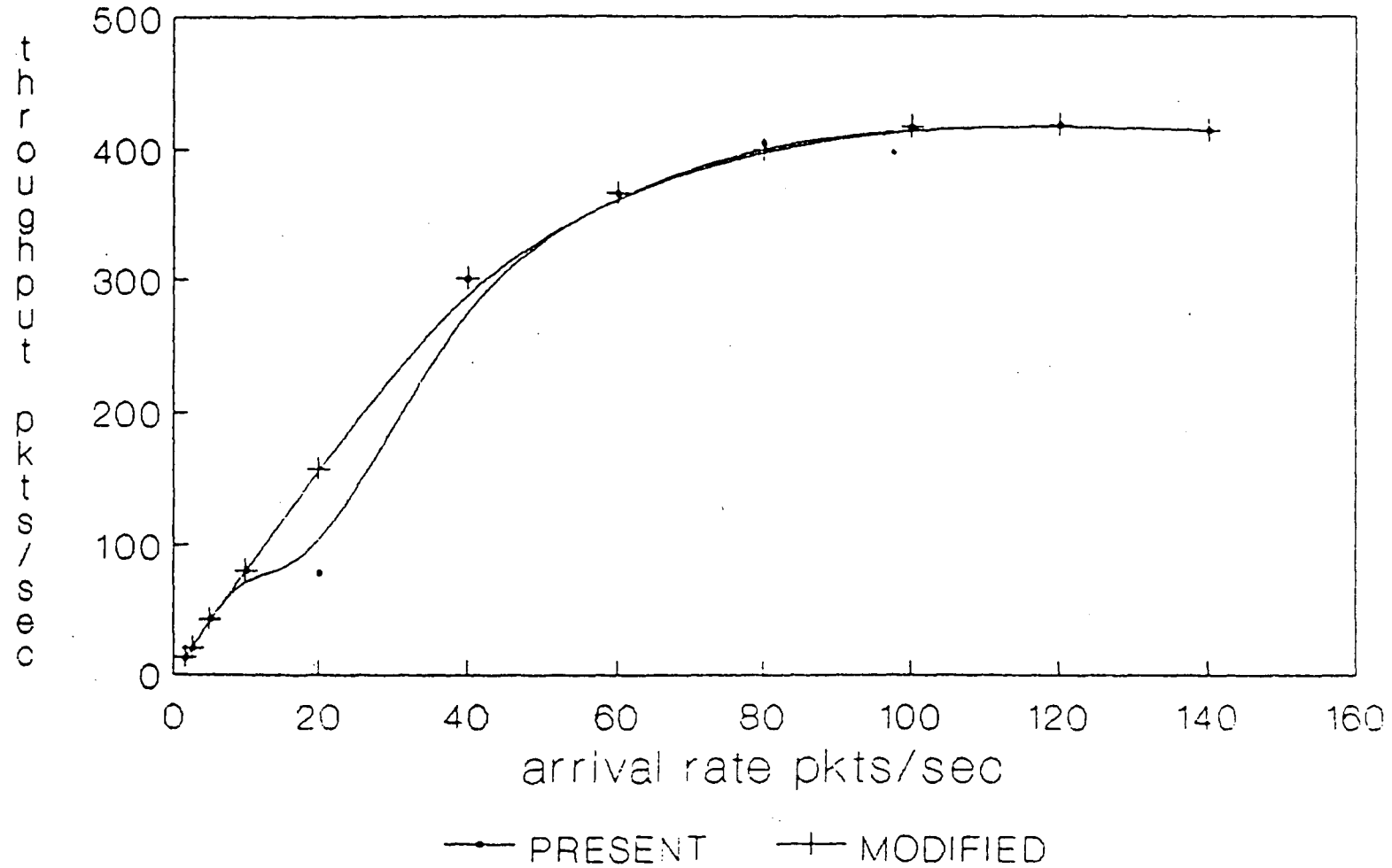
The delay comparison shown in fig. 3.2(b) for models 3 and 4 shows almost similar results, however model 4 shows better results (less delay) at low traffic than model 3

The throughput for models 1 and 2 is found to increase with arrival rate as noticed from fig 3.2(c) with model 2 showing better results (higher throughput) at low traffic.

The delay as observed from fig.3.2(d) is found to increase with arrival rate for both models. Model 2 however shows better performance, significantly at lower traffic.

It is thus observed that model 2 with the proposed cost function and modified routing algorithm gives the best results of the four models compared .

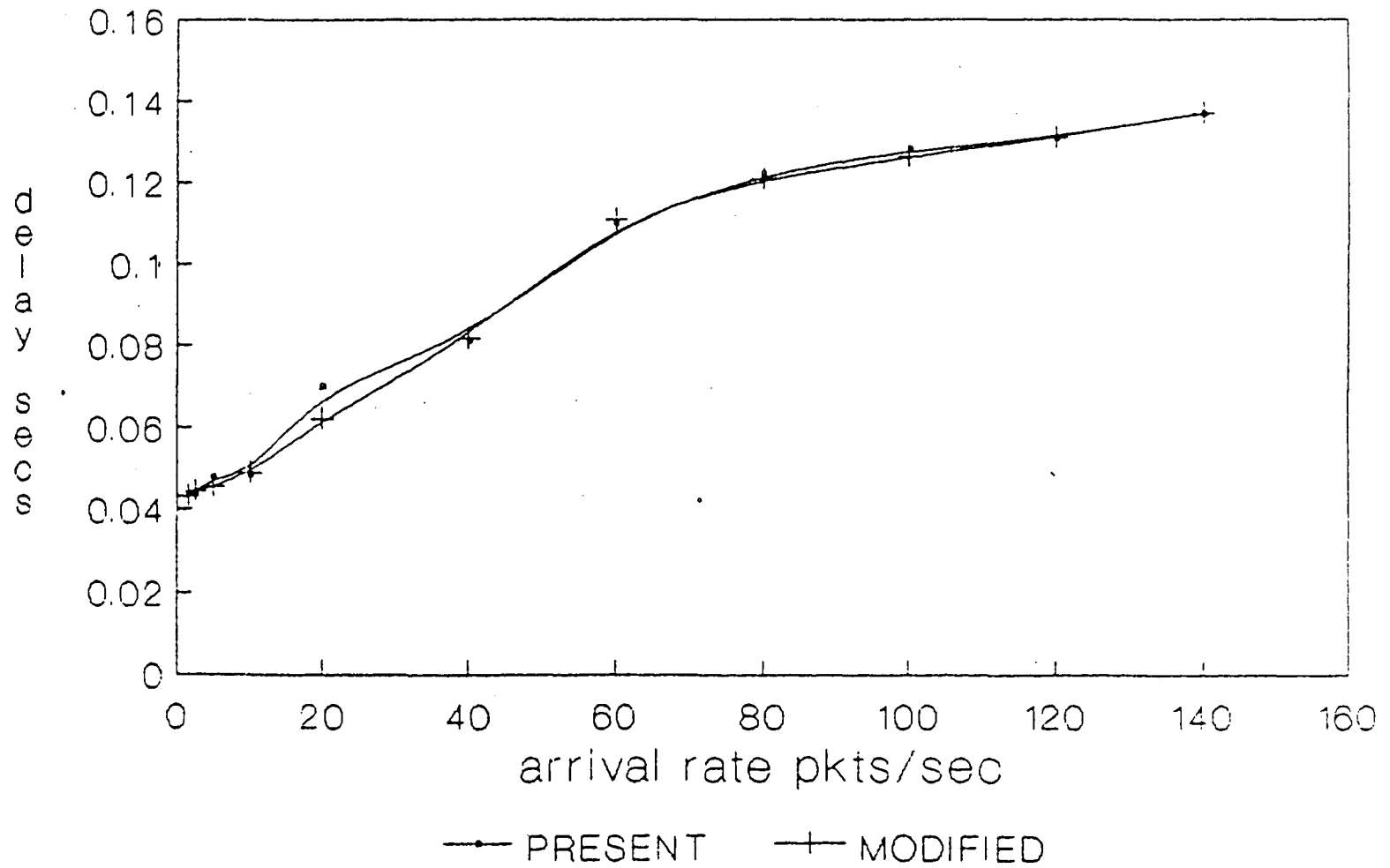
THROUGHPUT COMPARISON



Model 3 Vs Model 4

Fig 3.2 (a)

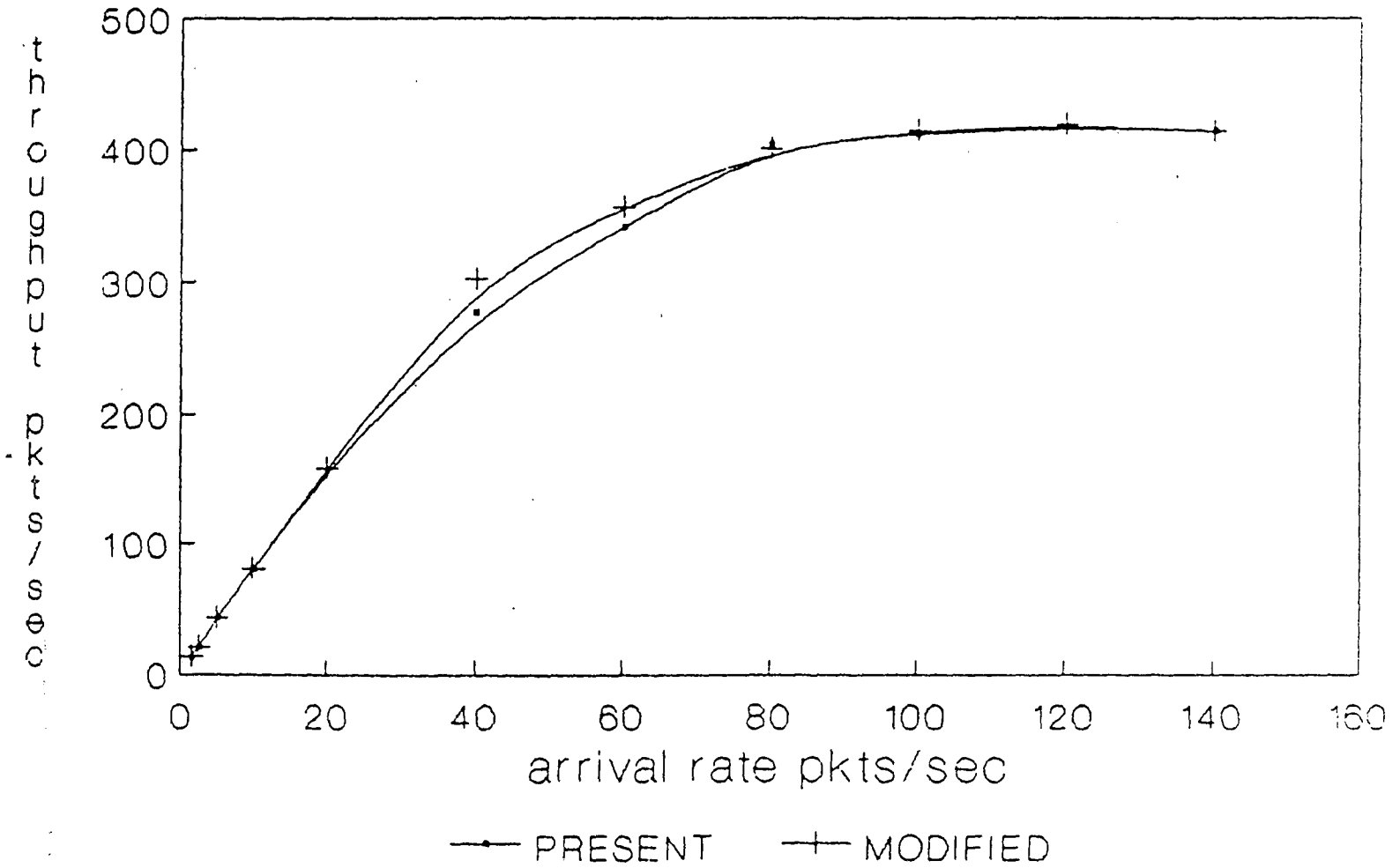
DELAY COMPARISON



Model 3 Vs Model 4

Fig 3.2 (b)

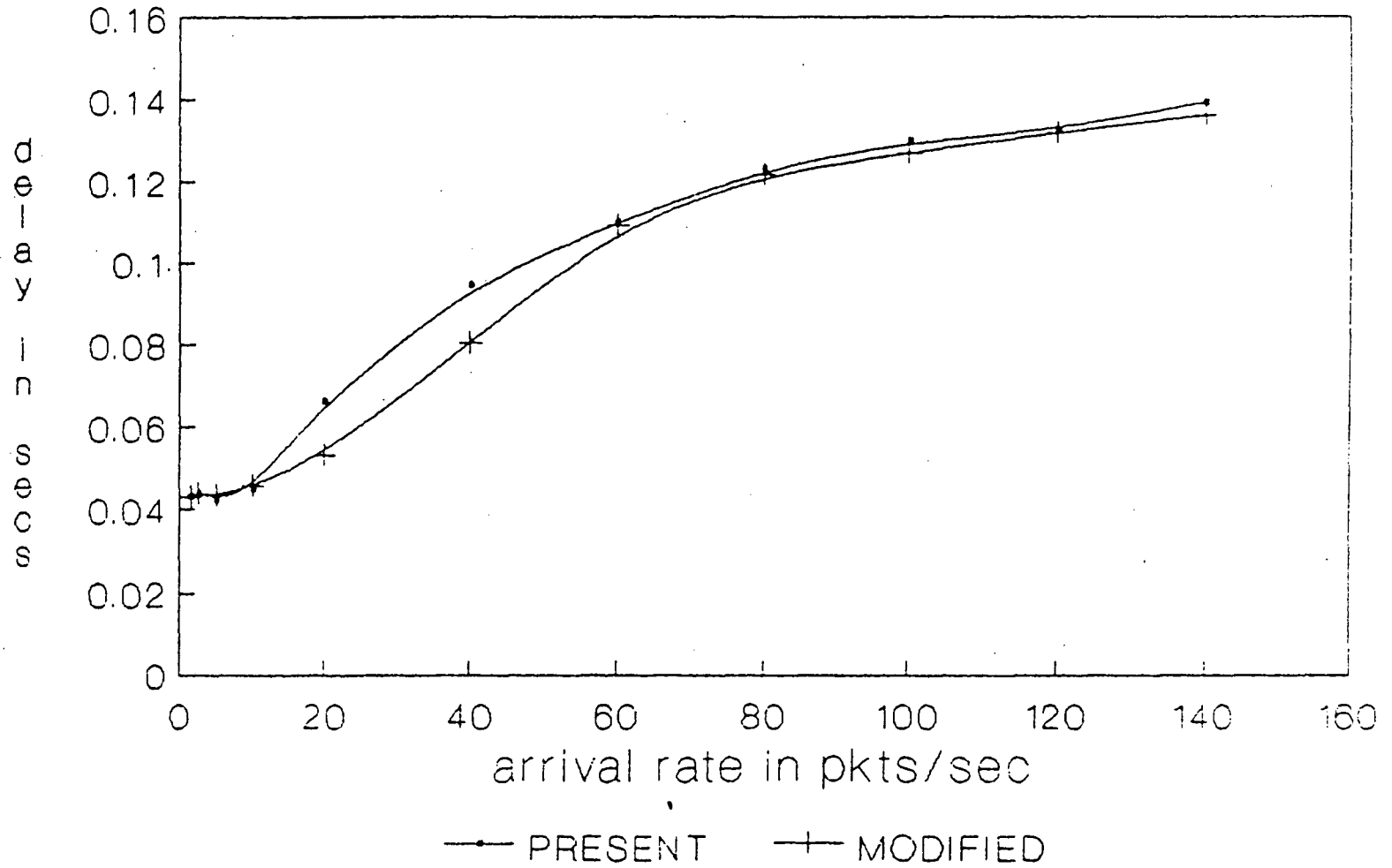
THROUGHPUT COMPARISON



Model 1 Vs Model 2

Fig 3.2 (c)

DELAY COMPARISON



Model 1 Vs Model 2

Fig 3.2 (d)

Model No.	Cost Metric	Routing Algorithm	Performance
1.	Time Delay	Shortest Path First	----
2.	Max. U_{ij} * No. of Hops	Modified Algorithm	Best Performance
3.	Sum of Link Utilizations	Shortest Path First	Better than 1 but less than 2
4.	Sum of Link Utilizations	Modified Algorithm	Better than 3 but less than 2

RELATIVE PERFORMANCE

Table 3.2

CHAPTER-IV

CONCLUSION

Rapid advances in the field of electronics and communication has made high speed data transmission possible. Computer Networks have kept pace with these advancements. Consequently the traffic carried by them has increased significantly. A major criterion for evaluation of the performance of a computer network is its traffic carrying capability (throughput) and the mean delay incurred by a packet to reach from a source to a destination. Routing plays a pivotal role in improving these performance measures.

In this dissertation an attempt has been made to improve these performance measures for ARPANET by using a cost function based on link utilization and a modified routing algorithm has been proposed.

The routing strategy suggested differs from the presently running algorithm on ARPANET in

- * the routing algorithm
- * on the basis for cost metric and
- * the cost function

Since the evaluation of the performance of an adaptive routing algorithm is a complex task, simulation is

relied on exclusively. Four simulation models with different combinations of cost function and routing algorithms were developed.

It was observed that the proposed cost function and modified routing algorithm gave much better performance than the presently running algorithm on ARPANET. The difference in performance was observed to be more prominent in the case of a six node fully connected network topology than an eight node mesh topology. Table 3.2 (sec 3.6) tabulates the performance of the four models comprehensively.

The improvement in performance can be attributed to the better correlation between routing and congestion. This is important because routing affects congestion as it decides which resources will be used to transport messages. A lack of co-ordination between the two can easily lead to the selection of a congested route, such an action if cumulative could lead to deterioration in system performance considerably. This has been overcome by using a cost function based on utilization.

On the basis of the study carried out and results obtained it is noticed that the proposed cost function and the modified routing algorithm suggested if used on ARPANET should yield better performance.

4.1 SCOPE FOR FURTHER INVESTIGATION :

The work can be extended further to the

- 1.. Study of the performance of the modified routing algorithm on a network with higher number of nodes and different topologies.

2. Study of the affect of congestion on the modified routing algorithm suggested.

3. Study of the performance of the suggested algorithm with variation in update methodologies.

APPENDIX - I

BIBLIOGRAPHY

1. DIMITRI BETSEKAS & ROBERT GALLAGER, **DATA NETWORKS**, Prentice hall, second edition, pp. 363-403.
2. E.C. ROSEN "THE UPDATING PROTOCOL OF ARPANET ROUTING ALGORITHM". Computer Networks 4, 1980, pp. 11-19.
3. ANDREW S. TANENBAUM & WISKUNDIG SEMINARIUM "NETWORK PROTOCOLS", Computing Survey Dec. 1981, pp. 191-196.
4. INDER M. SOI and KRISHAN K. AGGARWAL, "A REVIEW OF COMPUTER COMMUNICATION NETWORK CLASSIFICATION SCHEMES," IEEE Communications, March 1981, pp. 16-23.
5. MISCHA SCHWARTZ and THOMAS E. STERN, "ROUTING PROTOCOLS," pp. 327-343.
6. PARVIZ KERMANI and LEONARD KLEINROCK, "VIRTUAL-CUT-THROUGH: A NEW COMPUTER COMMUNICATION SWITCHING TECHNIQUE" Computer Networks 3, 1979.
7. LAWRENCE, G. ROBERTS, "THE EVOLUTION OF PACKET SWITCHING," IEEE, 1978.
8. ROBERT E. KAHN and WILLIAM R. CROWTHER, "FLOW-CONTROL IN RESOURCE SHARING COMPUTER NETWORK," IEEE 1971, pp. 539-546
9. JOHN Mc. QUILLAN, GILBERT FALM & IRA RICHER, "A REVIEW OF THE DEVELOPMENT AND PERFORMANCE OF ARPANET ROUTING ALGORITHM," IEEE 1978, pp. 1802-1811.
10. JOHN. M. QUILLAN, IRA RICHER & ERIC. C. ROSEN, "THE NEW ROUTING ALGORITHM FOR THE ARPANET," IEEE On Communications, May 1980, pp. 711-719.
11. RICHERS BARNETT & SALLY M. SMITH. **PACKET -SWITCHED NETWORKS**, Sigma Press, pp. 2-41.

12. ED. SHOEMAKER, **COMPUTER NETWORKS & SIMULATION I & II.**
13. TANENBAUM, **COMPUTER NETWORKS.**
14. WILSHOW CHOU, "COMPUTER COMMUNICATION NETWORKS : THE PARTS MAKE UP THE WHOLE," Proc. AFIPS Nat Computers Conference, May 1975.
15. D.W. GLAZER & C. TROPPER, "A NEW METRIC FOR DYNAMIC ROUTING ALGORITHM," IEEE Trans Communications, March 1990.
16. M. GERLA, "DETERMINISTIC AND ADAPTIVE ROUTING POLICIES IN PACKET SWITCHED COMPUTER NETWORK," IEEE Data communication system, November 1973.
17. FRANCIS NEELAMKAVIL, **COMPUTER SIMULATION & MODELLING,** John Willey & Sons, 1987.

APPENDIX - II

```

/* DEF.H */

/* DECLARATIONS */
/*-----*/
/*FILES TO BE INCLUDED */

#include <stdio.h>
#include <math.h>
/*-----*/
/* GLOBAL ARRAYS */

EXTERN float link[20][20];
EXTERN int destination[20][20];/** routing table **/
EXTERN int precede[20][20]; /** do **/
EXTERN int ccost[20][20]; /* indicates change in cost */
EXTERN int cpktrd[20][20][20];/* control pkt received num. */
EXTERN float capacity[20][20];/* of links in bits/sec */
EXTERN float lambda[20]; /* mean inter arrival rate *
EXTERN float cost[20][20]; /*cost of transmission on link*/
EXTERN float distance[20][20];/*distance of node from root node*/
EXTERN float ncost[20][20]; /* the changed cost calculated */
EXTERN float delta[20][20]; /*change in time delay permissible*/

/*-----*/
/* STRUCTURE DEFINITIONS */

EXTERN struct packet {

    struct packet *next;/**pointer to form link list*/
    float pkgntm;/**packet generation time */
    float inttm;/**intermediate time for arrival*/
    int sourcend;/** source node*/
    int dest;/**final destination node */
    int intsnd;/** intermediate source node */
    int intdnd;/**intermediate destination node */
    int cpktnu;/** control pkt num. */
    int inorexorc;/**internal=1,node =3,ext=0,cont=2*/
    int nhps;/** number of hops */

```

```

        }.*linkqueue[20][20][5] ;/*link queues */

EXTERN struct {
    int    linkbusy; /*busy =1,else =0*/
    float  pktdvd; /*no. of pkts delivered on link */
    int    f,r; /*front & rear pointers for linkqueue*/
    float  linkcmt; /* expected time for link to get free */
    float  trf; /*traffic in stipulated interval */
    float  tottrf; /*total traffic on link */
    float  delay; /* the time delay per link */
    float  intpktdvd; /* no. of pkts generated in cpgint */
    float  wttim; /*waiting time*/
    struct packet *cpkt; /*pointer to control packet link list */
        }llst[20][20];

EXTERN struct nslist {
    int father;
    int son ;
    float dist;
    struct strshp *next ;
} ;
EXTERN      struct {
    float maxul;
    float cost;
    int   nhps;
} dist[20][20] ;

/*-----*/

/*GLOBAL VARIABLES*/

EXTERN int nonds; /* number of nodes */
EXTERN float mainclock;
EXTERN float pktsize; /*variable packet size in bits*/
EXTERN float cpktsize ; /* control pkt size */
EXTERN float mgsrejected; /* number of external mgs rejected */
EXTERN float imgsrejected; /*number of internal mgs rejected */

```

```

EXTERN float mgsdvd; /*number of mgs delivered */
EXTERN float cmgdvd; /*no. of control mgs delivered */
EXTERN float mgsgen; /*number of mgs generated */
EXTERN int routal; /*routing algorithm number */
EXTERN float simtim; /*simulation time */
EXTERN float timnk; /*time spent by pkts delivered in network */
EXTERN float cpgint ; /*control pkt gen interval */
EXTERN float cpkgtim ; /* control pkt gen time */
EXTERN float updtim ; /*update time after cpkt gen */
EXTERN struct packet *list; /*pointer to event list */
EXTERN int tnhrs ; /* total no. of hops */
EXTERN int cpktnum ; /* control pkt num. */
EXTERN float genrand();
/*-----*/
/* GLOBAL CONSTANTS */

#define MAXNODE 20
#define QUEUES 5
#define TRUE 1
#define FALSE 0
#define PKTSZ 80
#define NUL 0
#define INFINITY 999
#define MEMBER 1
#define NONMEMBER 0
#define new(p)p=(struct packet *) malloc(sizeof(struct packet))
#define aloc(p) p=(struct nslst *) malloc(sizeof(struct nslst))
/*-----*/

```

```

/*-----*/
/*MODIFIED ROUTING ALGORITHM*/
/*-----*/
#define EXTERN extern
#include"def.h"

/** To find the maximum of two given floating point numbers **/

float max(a,b)
    float a,b;
    {
        float c;
        c = ( b>a ) ? b : a ;
        return(c);
    }
/*-----*/
/** If two paths are of equal cost --- used in reroute1 **
int equal(l,i,j)
    int l,i,j;
    {
        float u;
        u = max(dist[l][i].maxul , cost[i][j]);
        if ( u*(dist[l][i].nhps + 1) == dist[l][j].cost)
            if ( u > 0.8 )
                {
                    if (dist[l][i].maxul > dist[l][j].maxul)
                        return(0);
                    else return(1);
                }
            else {
                if (( dist[l][i].nhps + 1 ) > dist[l][j].nhps )
                    return(0);
                else return(1);
            }
        else return(0);
    }
/*-----*/

```

```
/* Shortest path algorithm modified for uil*nhps model */
```

```
int allmem(x)
```

```
int x[];
```

```
{
```

```
int i = -1;
```

```
do {
```

```
    i=i+1;} while(i<nonds && (x[i] == 1));
```

```
if (i < (nonds -1 )) return(FALSE);
```

```
else return(TRUE);
```

```
}
```

```
void shpdijk(s)
```

```
int s;
```

```
{
```

```
int current,i,k;
```

```
int perm[20] ;
```

```
float dc,smalldist,newdist;
```

```
/* Initialization */
```

```
for(i=0; i<20; i++){
```

```
perm[i] = NONMEMBER ;
```

```
dist[s][i].maxul = 0 ;
```

```
dist[s][i].cost = INFINITY ;
```

```
dist[s][i].nhps = 0 ;
```

```
precede[s][i]= -1;
```

```
}/* end for */
```

```
perm[s] = MEMBER;
```

```
dist[s][s].cost = 0;
```

```
current = s;
```

```
while (allmem(perm) == FALSE )
```

```
{
```

```
    smalldist = INFINITY ;
```

```
    dc = dist[s][current].cost;
```



```

for(i=0;i<nonds ; i++)
if ( perm[i] == NONMEMBER )
{ newdist = max(dist[s][current].maxul ,
                cost[current][i])*(dist[s][current].nhps);
  if (newdist < dist[s][i].cost)
  {
/* distance from s to i through current is smaller than
                                dist[s][i].cost */
    dist[s][i].cost = newdist;
    dist[s][i].maxul= max(dist[s][current].maxul ,
                          cost[current][i]);
    dist[s][i].nhps = dist[s][current].nhps + 1 ;
    precede[s][i] = current;
  } /* end if */
/* determine the smallest distance */
  if ( dist[s][i].cost < smalldist )
  {
    smalldist = dist[s][i].cost ;
    k = i;
  } /* end if */

  } /* end for --- if */

current = k;
perm[current] = MEMBER ;

  } /* end while */
return;
}

/* Precede matrix is now used to generate the routing table at node

/* void destb()
{
int i,s,d1,d2;
for(s=0;s<nonds;s++)

```

```

        for(i=0 ; i < nonds ; i++)
            if( i!= s)
    {
        d2 = precede[s][i];
        if ( d2 == s ) destination[s][i] = i ;
        else {while(d2 != s)
            {
                d1 = d2 ;
                d2 = precede[s][d1] ;
            }

            destination[s][i] = d1 ;
        }
    }

    return;
}*/

/*-----*/
/* reroute() used to update the minimum spanning tree - shortest path
first algorithm */

void reroute2()
{
    struct nslist *np1 , *np2 , *np3 ,*np4 ;
    float delta,del;
    int i,j,k,l,m,current,n,o,flag,flag1;
    int subtree[20];
    struct nslist* addnp1();
    for( i = 0 ; i < nonds ; i++)
        for ( j =0 ; j< nonds ; j++)
            if (( ccost[i][j] == 1 ) && ( i !=j))
                {
                    del = ncost[i][j] - cost[i][j] ;
                    cost[i][j] = ncost[i][j];
                    ccost[i][j] = -1;
                    for(l = 0 ; l < nonds ; l++)
                        {
                            np1 = NUL ;

```

```

        if (precede [l][j] == i)
            delta = del ;
        else
        { delta = distance[l][i] + cost[i][j] - distance[l][j];
          if ( delta >= 0 ) continue ;
          else if ( precede[l][i] != j )
              precede[l][j] = i ;
          else continue ;
        }
    dist[l][j].maxul = max(dist[l][i].maxul , cost[i][j]);
    dist[l][j].cost = dist[l][j].maxul*(dist[l][i].nhps + 1 );
    dist[l][j].nhps = dist[l][i].nhps + 1 ;

    for( m = 0 ; m < 20; m++)
        subtree[m] = -1 ;
    k= 0 ;
    for(m=0;m<nonds;m++)
    if( precede[l][m] == j )
        { subtree[k] = m ;
          dist[l][m].maxul = max(dist[l][j].maxul , cost[j][m]);
          dist[l][m].nhps = dist[l][j].nhps + 1 ;
          dist[l][m].cost = dist[l][m].maxul * dist[l][m].nhps ;
          k++ ;
        }

        current = 0 ;
    while ( current < k)
        { for (m=0; m < nonds;m++)
          if ( precede[l][m] == subtree[current])
              { flag =1;
                for( n =0 ; n < k ;n++)
                    if (subtree[n] == m ) flag =0;
                if(flag == 1)
                    { subtree[k] = m;
                      dist[l][m].maxul =
max(dist[l][subtree[current]].maxul , cost[subtree[current]][m])

```

```

dist[l][m].nhps = dist[l][subtree[current]].nhps + 1 ;
dist[l][m].cost = dist[l][m].maxul * dist[l][m].nhps ;
k++;
    }
}
current++;
} /* end while */

if ( delta > 0 )
for(m=0; m<k ; m++)
for( n=0 ; n < nonds ; n++)
if( link[n][subtree[m]] > 0)
if( (max(dist[l][n].maxul ,
cost[n][subtree[m]])*(dist[l][n].nhps + 1)) <
dist[l][subtree[m]].cost
/* n should not be in the subtree */
{ flag = 0 ;
for( o = 0 ; o < k ; o ++ )
if ( subtree[o] == n ) flag = 1;
if ( flag == 0 )
{ alloc(np2);
np2->father = n ;
np2->son = subtree[m] ;
np2->dist =(max(dist[l][n].maxul , cost[n][subtree[m]])
*(dist[l][n].nhps + 1 ) ;
np2->next = NUL;
np1=addnp1(np1,np2);
}
}
if ( delta < 0)
for(m=0; m<k ; m++)
for( n=0 ; n < nonds ; n++)
if( link[subtree[m]][n] >0 )
if (max( dist[l][subtree[m]].maxul , cost[subtree[m]][n])*
(dist[l][subtree[m]].nhps + 1)<dist[l][m].cost)
/* n should not be in the subtree */

```

```

{
    flag = 0 ;
    for( o = 0 ; o < k ; o ++ )
        if ( subtree[o] == n ) flag = 1;
    if ( flag == 0 )
    {
        aloc(np2);
        np2->father = subtree[m];
        np2->son = n ;
        np2->dist = (max( dist[l][subtree[m]].maxul
            ,cost[subtree[m]][n])*(dist[l][subtree[m]].nhps + 1)
        np2->next = NUL;
        np1=addnp1(np1,np2);
    }
}
while ( np1 != NUL )
{
    np2 =np1;
    np1 = np1->next;
    np2->next = NUL ;
    if (( dist[l][np2->son].cost < np2->dist )||(np2->son == 1)||
        (precede[l][np2->father] == np2->son ))
        free((char *)np2);
    else {
        precede[l][np2->son] = np2->father;
        dist[l][np2->son].cost = np2->dist;
        dist[l][np2->son].maxul= max(dist[l][np2->father].maxul ,
            cost[np2->father][np2->son])
        dist[l][np2->son].nhps = dist[l][np2->father].nhps + 1 ;
        /* Check whether any neighbours of this node added can be
        rerouted. If the neighbour is already present in the shortest
        path tree checking */
        for(m=0;m<nonds;m++)
            if ( link[np2->son][m] >0)
            {
                np3 = np4=np1 ;
                while ((np3 != NUL) && ( np3->son != m))
                {

```

```

np4 = np3;
np3 = np3->next;
}
if(np3 == NUL)
  { if (max( dist[l][np2->son].maxul , cost[np2->son][m])*
      (dist[l][np2->son].nhps + 1) < dist[l][m].cost)
    {
      aloc(np3);
      np3->son = m ;
      np3->father = np2->son ;
      np3->dist = (max(dist[l][np2->son].maxul , cost[np2->son][m])*
                  (dist[l][np2->son].nhps + 1 ) ;
      np3->next = NUL ;
      np1=addnp1(np1,np3);
    }
  }
else {
  if (max( dist[l][np2->son].maxul , cost[np2->son][m])*
      (dist[l][np2->son].nhps + 1) < dist[l][m].cost )
    { if ( np1 == np3) np1=np1->next;
      np4->next = np3->next ;
      np3->father = np2->son ;
      np3->dist= np2->dist + cost[np2->son][m];
      np3->next = NUL;
      np1=addnp1(np1,np3);
    }
  }
}
free((char *)np2);
}/* end while (np1!=NUL)*/
}
}

```

```

/*-----*/
/* Present Arpanet Algo */
/*-----*/
#defineEXTERNextern
#include "def.h"
/*-----*/

/* reroute() used to update the minimum spanning tree - shortest
path first algorithm */

void reroute()
{
struct nslist *np1 , *np2 , *np3 ,*np4 ;
float delta,del;
int i,j,k,l,m,current,n,o,flag,flag1;
int subtree[20];
struct nslist* addnp1();

for( i = 0 ; i < nonds ; i++)
  for ( j =0 ; j< nonds ; j++)
    if (( ccost[i][j] == 1 ) && ( i !=j))
      {
        del = ncost[i][j] - cost[i][j] ;
        cost[i][j] = ncost[i][j];
        ccost[i][j] = -1;
        for(l = 0 ; l < nonds ; l++)
          {
            np1= NUL ;
            if (precede [l][j] == i)
              { delta = del ;
                distance[l][j] += delta ;
              }
            else
              { delta = distance[l][i] + cost[i][j] - distance[l][j];
                if ( delta >= 0 ) continue ;
                else if ( precede[l][i] != j ) {
                  precede[l][j] = i ;
                }
              }
          }
      }
}

```

```

        distance[l][j] = distance[l][i] + cost[i][j] ;
    }
    else continue ;
}
for( m = 0 ; m < 20; m++)
    subtree[m] = -1 ;
k= 0 ;
for(m=0;m<nonds;m++)
if( precede[l][m] == j )
    { subtree[k] = m ;
      k++ ;
    }
    current = 0 ;
while ( current < k)
{ for (m=0; m < nonds;m++)
    if ( precede[l][m] == subtree[current])
    { flag =1;
      for( n =0 ; n < k ;n++)
      if (subtree[n] == m ) flag =0;
      if(flag == 1)
      { subtree[k] = m;
        k++;
      }
    }
    current++;
} /* end while */

for( m = 0 ; m < k ;m++)
distance[l][subtree[m]] += delta;
if ( delta > 0 )
for(m=0; m<k ; m++)
for( n=0 ; n < nonds ; n++)
if( link[n][subtree[m]] > 0)
if (( distance[l][n] + cost[n][subtree[m]]) <
                                     distance[l][subtree[m]])
/* n should not be in the subtree */
{ flag = 0 ;
                                     /*Vexis.c*/

```



```
/*Vexis.c*/
```

```

    for( o = 0 ; o < k ; o ++ )
    if ( subtree[o] == n ) flag = 1;
    if ( flag == 0 )
    { aloc(np2);
      np2->father = n ;
      np2->son = subtree[m] ;
      np2->dist = distance[l][n] + cost[n][subtree[m]] ;
      np2->next = NUL;
      np1=addnp1(np1,np2);
    }
  }
}

```

```

if ( delta < 0 )
  for(m=0; m<k ; m++)
  for( n=0 ; n < nonds ; n++)
  if( link[subtree[m]][n] >0 )
  if ( ( distance[l][subtree[m]] + cost[subtree[m]][n]) <
                                             distance[l][n])

```

```
/* n should not be in the subtree */
```

```

{
  flag = 0 ;
  for( o = 0 ; o < k ; o ++ )
  if ( subtree[o] == n ) flag = 1;
  if ( flag == 0 )
  { aloc(np2);
    np2->father = subtree[m];
    np2->son = n ;
    np2->dist = distance[l][subtree[m]] + cost[subtree[m]][n] ;
    np2->next = NUL;
    np1=addnp1(np1,np2);
  }
}

```

```

while ( np1 != NUL )
{ np2 =np1;
  np1 = np1->next;
}

```

```

np2->next = NUL ;
if (( distance[1][np2->son] < np2->dist )||(np2->son ==
        1)||(precede[1][np2->father] == np2->son))
    free((char *)np2);
else {
    precede[1][np2->son] = np2->father;
    distance[1][np2->son] = np2->dist;
/* Check whether any neighbours of this node added can be
rerouted. If the neighbour is already present in the shortest
path tree checking */
    for(m=0;m<nonds;m++)
    if ( link[np2->son][m] >0)
    { np3 = np4=np1 ;
      while ((np3 != NUL) && ( np3->son != m))
      {
          np4 = np3;
          np3 = np3->next;
      }
      if(np3 == NUL)
      { if (( distance[1][np2->son] + cost[np2->son][m])
          < distance[1][m] )
          {
              aloc(np3);
              np3->son = m ;
              np3->father = np2->son ;
              np3->dist = np2->dist + cost[np2->son][m];
              np3->next = NUL ;
              np1=addnp1(np1,np3);
          }
      }
    }
    else {
        if ((np2->dist + cost[np2->son][m]) < np3->dist)
        { if( np1 == np3) np1 = np1 ->next;
          np4->next = np3->next ;
          np3->father = np2->son ;
        }
    }
}

```

```

    np3->dist= np2->dist + cost[np2->son][m];    /*Vexis.c*/
    np3->next = NUL;
    np1=addnp1(np1,np3);
  }
}
}
}
free((char *)np2);
}/* end while (np1!=NUL)*/
}
}

```

/* addnp1() adds the node in the list with starting pointer np1 in the increasing order of their distances */

```

struct nslist* addnp1(np1,np2)
struct nslist *np1,*np2 ;
{
  struct nslist *np3,*np4;
  if( np1 == NUL ) np1 = np2;
  else {
    np3 = np1;
    if ( np1->dist >= np2->dist)
      { np2->next = np1;
        np1 = np2 ;
      }
    else {
      while ((np3->dist < np2->dist ) && (np3!= NUL))
        {
          np4=np3;
          np3 = np3->next;
        }
      if(np3 == NUL){
        np2->next = NUL ;
        np4->next = np2 ;
      }
    }
  }
  else{

```


18

```

/*-----*/
/* MAIN SIMULATION MODULE */
/*-----*/
#defineEXTERN
#include "def.h"

/* Initialisation of various data structures and costants used */
/* for restarting the entire simulation .*/

void initialise()
{
    void varinit();
    int i,j;
    for (i=0;i<20 ;i++)
        for (j=0;j<20;j++)
        {
            link[i][j] = -1;
            capacity[i][j] = 0.0;
            lambda[j] = 0.0;
        }
    nonds = 0;
    pktsize = 0;
    routal = 0;
    simtim = 0.0;
    cpgint = 0.0;
    varinit();
}

/* Initialising variables not data read */
void varinit ()
{
    int k,l,m;
    struct packet *nptr,*ndp;
    for (k=0;k<20;k++)
    for(l=0;l<20;l++ )
    {
        llst[k][l].linkbusy = 0;
    }
}

```

```
llst[k][1].linkcmpt = 0.0;
llst[k][1].tottrf=0.0;
llst[k][1].trf=0.0;
llst[k][1].pktdvd=0;
llst[k][1].wttim=0.0;
llst[k][1].f=0;
llst[k][1].r=0;
llst[k][1].delay = 0.0 ;
llst[k][1].intpktvd = 0.0;
```

```
ncost[k][1] = 999.0;
delta[k][1] = 999.0;
cost[k][1] = 999.0;
ccost[k][1] = 0 ;
destination[k][1] = 0 ;
precede[k][1] = -1;
```

```
nptr = llst[k][1].cpkt;
llst[k][1].cpkt=NUL;
while (nptr != NUL)
{
    ndp = nptr;
    nptr = nptr->next;
    free((char *)ndp);
}
for ( m= 0 ;m<20;m++)
    cpktrd[k][1][m] = -1 ;
for (m=0;m<5;m++)
{
    nptr=linkqueue[k][1][m];
    linkqueue[k][1][m]=NUL;
    while(nptr!=NUL)
    { ndp=nptr;
      nptr=nptr->next;
      free((char *)ndp);
    }
}
```

```

    }
    if (list == NUL)
        {new(list);
         list->next=NUL;
         list->inorexorc = 0;
         list->inttm=9999.0;
         }
    else {
        while.(list->next != NUL )
            {
                nptr=list;
                list=list->next;
                free((char *)nptr);
            }
    }

    /** Initialising Global Variables **/

    mainclock =0.0;
    mgsrejected =0;
    imgsrejected =0;
    mgsdvd=0;
    cmgdvd=0;
    cpkgtim = 0.0 ;
    cpktnum = 0 ;
    mgsgen=0;
    timnk=0.0;
    tnhps=0;

    }

    /*-----*/
    /* Reads input : Network config , Link Capacities and          */
    void readinput()
    { int k,l,flag;
      while ((nonds == 0)!!(nonds >20 ))
          { printf(" Enter the number of nodes in the Network.");

```

```

printf(" (Max. no. of permissible nodes is %d )",MAXNODE);
printf(" Enter :");
scanf("%d",&nonds);}
for (k=0; k<nonds;k++)
  { for (l=0 ; l < nonds ;l++ )
    {if ( k == l)
      {
        link[k][l] = 0;
        capacity[k][l] = 0 ;
      }
    else if(k>l)
      { link[k][l] = link[l][k];
        capacity[k][l] = capacity[l][k];
      }
    else {
      if (link[k][l]== 1 ;!link[k][l]==0) flag =TRUE ;
      else flag = FALSE;
    }
    while(!flag)
    { printf(" Enter 1 if link %d %d is present else 0 :",k,l);
      scanf("%f",&link[k][l]);
      if(link[k][l]==1 ;! link[k][l]==0) flag =TRUE;
    }
    while (link[k][l] == 1 && capacity [k][l]<=0)
    {
      printf(" Enter the capacity of links in bits/sec (>0) : ");
      scanf( "%f",&capacity[k][l]);
    }
  }
}
}
for( k = 0 ; k < nonds; k++)
{
printf(" Enter the mean interarrival rate (pkt/sec) lambda at
                                             each node .");
printf(" Enter lambda for node %d : ",k);
scanf("%f",&lambda[k]);
}

```



```

printf(" Enter packet size (in bits) : ");
/*packet size in bits if capacity of channel is in bits/sec *
scanf("%f",&pktsize);

/*Reading the simulation period & updating interval */
printf(" Enter the period for which simulation is to be carried
                                         out in secs: ");
scanf("%f",&simtim);
printf(" Enter the UPDATING INTERVAL in secs : ");
scanf ("%f",&cpgint);
}

/*-----*/

/* To check whether the given network configuration is connected
or not */
int connectedornot()
{
    int f,j,i,l=0,k= -1,flag=0;
    int fvt[20],svt[20];

    for(i=0 ; i< 20 ;i++)
    {
        fvt[i] = 0 ;
        svt[i] = 0 ;
    }
    i = 0;
    for(j=1;j<nonds;j++)
    if (link[i][j]==1)
    {
        fvt[j]=1;
        k++;
        svt[k]=j;
        f=1;
    }
    if (f==1) fvt[0]=1;

```

```

while (l<=k)
{
    i=svt[l];
    l++;
    for(j=0;j<nonds;j++)
    if((link[i][j]==1)&&(fvt[j]!=1))
    {fvt[j]=1;
    k++;
    svk[k]=j;
    }
}
for(i=0;i<nonds;i++)
{
    if(fvt[i] != 1)
    {
        printf(" Network is not connected .");
        flag=1;
        break;
    }
}
return(flag);
}
/* In the main prg if flag =1 read network config again */

/*-----*/
/* Random number generation in an interval [0,1] , rand() gives a
random number normalised in the interval 0,1 */
float genrand ()
{
    float x;
    do{ x=(float) rand()/32767.0 ;
    }while(x== 0.0);
    return(x);
}

/*-----*/

```

```

/* Inserts packets in event list */
/* Insert function including the cpkt gen and updating nodes in
event list */
void loop1(nptr,nrf,nrf1)
    struct packet *nptr,*nrf,*nrf1;
    { void inst (nptr,nrf,nrf1);
      nrf1= nrf;
      nrf= nrf->next;
      inst(nptr,nrf,nrf1);
      return;
    }

void loop2(nptr,nrf,nrf1)
    struct packet *nptr, *nrf , *nrf1;
    {
      nptr->next = nrf;
      nrf1->next = nptr;
      return;
    }

void inst(nptr,nrf,nrf1)
    struct packet *nptr,*nrf,*nrf1;
    {
      if (nrf->inttm > nptr->inttm ;; nrf->next == NUL )
      { nptr->next = nrf;
        nrf1->next = nptr;
        return;
      }
      else if (nrf->inttm == nptr->inttm)
        switch (nptr->inorexorc)
        {
          case 5 : loop2(nptr,nrf,nrf1);
                  return;
          case 4 : if(nrf->inorexorc < 4)
                  {

```

```

        loop2(nptr,nrf,nrf1);
        return;
    }
    else {
        loop1(nptr,nrf,nrf1);
        return;
    }
case 3 : loop1(nptr,nrf,nrf1);
        return;
case 2 : if ( nrf->inorexorc > 3 ;;
            nrf->inorexorc== 2)
        { loop1(nptr,nrf,nrf1);
          return;
        }
        else loop2(nptr,nrf,nrf1);
          return;
case 1 : if ( nrf->inorexorc != 3 )
        { loop1(nptr,nrf,nrf1);
          return;
        }
        else loop2(nptr,nrf,nrf1);
          return;
    }
}

void insert(nptr)
    struct packet *nptr;
{
    struct packet *nrf,*nrf1 ;

    nrf=nrf1=list;
    if(nrf->inttm > nptr ->inttm)
        { list = nptr;
          nptr->next = nrf;
          return;
        }
    while ((nrf->next != NUL ) && (nrf->inttm < nptr->inttm))

```

```

    {
        nrf1 = nrf ;
        nrf = nrf->next ;
    }
    inst(nptr,nrf,nrf1);
    return;
}

/*-----*/
/**Initial generation of event list **/
void genevlst()
{ struct packet *nptr;
  int i;
  float ft;

  for(i=0;i<nonds;i++)
  { new(nptr);
    nptr->sourcend = i;
    nptr->inorexorc = 3 ;
    nptr->inttm = - ( 1.0/lambda[i])*(float)log(genrand());
    insert(nptr);
    nptr=NUL;
  }
  /* Finding the last time assigned to the event in list */
  nptr = list;
  do { ft = nptr->inttm;
      nptr = nptr-> next;
      }while (nptr->next != NUL);
  /* generating cpkt gen and update nodes */
  new(nptr);
  nptr->inorexorc = 4 ;
  nptr->inttm = ft + cpgint ;
  insert(nptr);
  new(nptr);
  nptr->inorexorc = 5 ;
  nptr->inttm = ft + cpgint + updtim;
  insert(nptr);
  nptr = NUL ;
}

```

```

    return;
}

/*-----*/
/* To get next event to be performed */
struct packet *getevent()
{
    struct packet *nptr, *ndp;
    int destin;
    ndp=list;
    if(list->next==NUL)
    { printf(" No event in list . ");
      return(NUL);
    }
    else
    { list =list->next;
      ndp->next =NUL;
      if (ndp->inorexorc != 3 )
          return(ndp);
      else {
          mgsgen++;
          new(nptr);
          nptr->next = NUL ;
          nptr->pkgntm = ndp->inttm;
          nptr->inorexorc=0;
          nptr->nhps =0;
          nptr->sourcend=ndp->sourcend;
          nptr->intsnd=ndp->sourcend;
          nptr->inttm =ndp->inttm;
          do {
              destin=(int)(genrand()*nonds);
          }while(destin >= nonds ;! destin== ndp->sourcend);
          nptr->dest = destin;
          ndp->inttm += -(1.0/lambda[ndp->sourcend])*log(genrand());
          insert(ndp);
          return(nptr);
      }
    }
}

```

```

        }
    }
}

/*-----*/
/* If queue is empty the packet is directly put into link list */
void departlf(nptr)
    struct packet *nptr;
{
    int i,j ;
    float sizepkt ;
    i = nptr->intsnd;
    j = nptr->intdnd;
    if (nptr-> inorexorc == 1 )
        sizepkt = pktsize ;
    else sizepkt = cpktsize ;
    llst[i][j].linkbusy = 1;
    llst[i][j].trf += sizepkt/capacity[i][j];
    llst[i][j].tottrf += sizepkt/capacity[i][j];
    nptr->inttm = mainclock + sizepkt/capacity[i][j];
    nptr->nhps++;
    llst[i][j].linkcmpt = nptr->inttm;
    llst[i][j].pktdvd++;
    llst[i][j].delay += sizepkt/capacity[i][j];
    llst[i][j].intpktdvd++;
    insert(nptr);
    return;
}

/*-----*/
/* Control Packet Generation */
void cpkgen()
{
    int gf;
    int i,j;
    struct packet *nptr ;
    struct packet *cpktrout(nptr);
    /* Updating the cost matrix */
    for( i =0 ; i<nonds ; i++)
        for( j =0 ; j <nonds ; j++)

```

```

if ( link[i][j] == 1 )
{
    ncost[i][j] = llst[i][j].trf / cpgint ;
    llst[i][j].trf = 0 ;
    llst[i][j].delay = 0.0;
    llst[i][j].intpkt dvd = 0.0 ;
}
    cpktnum++;
/* Generating the pkts & transmitting / putting in queue */
for(i=0; i<nonds ;i++)
for (j=0;j<nonds;j++)
    if (link[i][j] == 1)
        {
            if((float)fabs(ncost[i][j] - cost[i][j] ) < delta[i][j] )
            {
                delta[i][j] -= 0.0128 ;
            }
            else
            {
                delta[i][j] = 0.064 ;
                ccost[i][j] = 1 ;
                new(nptr);
                nptr->sourcend = i;
                nptr->dest = j ;
                nptr->pkgntm = mainclock ;
                nptr->intdnd = i ;
                nptr->intsnd = i ;
                nptr->inorexorc = 2 ;
                nptr->cpktnu = cpktnum;
                nptr->nhps = 0;
                nptr = cpktrout(nptr);
                free((char *)nptr);
            }
        }
        else continue ;
return;
}
/*-----*/

```



```

/* Routing of control packets */
struct packet *cpktrout(nptr)
    struct packet *nptr;
{
    int s,j,k,l;
    struct packet *npd,*cnp;
    s = nptr->intdnd;
    if(cpktrd[s][nptr->sourcend][nptr->dest] >= nptr->cpktnu)
        return(nptr);
    else {
        cpktrd[s][nptr->sourcend][nptr->dest] = nptr->cpktnu ;
        for(j=0;j<nonds;j++)
            if(link[s][j] == 1 )
                {
                    new(npd);
                    npd->sourcend=nptr->sourcend;
                    npd->intsnd = s ;
                    npd->dest = j ;
                    npd->intdnd = j;
                    npd->pkgntm = mainclock;
                    npd->inorexorc = 2;
                    npd->nhps = 0;
                    npd->next = NUL;
                    npd->cpktnu = nptr->cpktnu ;
                    k = s ;
                    l = j ;
                    if ( llst[k][l].linkbusy == 0 )
                        departlf(npd);
                    else {
                        cnp = llst[k][l].cpkt ;
                        if (cnp == NUL)
                            { cnp = npd;
                                llst[k][l].cpkt = cnp ;
                            }
                        else { while ( cnp->next != NUL)
                                cnp = cnp->next;
                                cnp->next = npd;
                            }
                    }
                }
    }
}

```

```

    }
    }
    npd = NUL;
    cnp = NUL ;
    }
    return(nptr);
    }
}

/*-----*/
/* Departure of control packets maintained as link list by
pointer in llst[][] array */
int cpktdp(i,j)
int i,j;
{ int flag = 0 ;
  struct packet *cnpt= NUL;
  if ( llst[i][j].cpkt == NUL )
    return(flag);
  else
  {
    cnpt = llst[i][j].cpkt;
    llst[i][j].cpkt = cnpt->next ;
    cnpt->next = NUL;
    llst[i][j].wttim = mainclock - cnpt->inttm;
    departlf(cnpt);
    flag = 1 ;
    return(flag);
  }
}

/*-----*/
/* Shortest path algorithm */
void shpdijk1(s)
int s;
{
  int current,i,k;
  int perm[20] ;
  float dc,smalldist,newdist;

```

```

/* Initialization */
for(i=0; i<20; i++){
perm[i] = NONMEMBER ;
distance[s][i] = INFINITY ;
precede[s][i]= -1;
}/* end for */
perm[s] = MEMBER;
distance[s][s] = 0;
current = s;
while (allmem(perm) == FALSE )
{
    smalldist = INFINITY ;
    dc = distance[s][current];
    for(i=0;i< nonds ; i++)
    if ( perm[i] == NONMEMBER )
    { newdist = dc + cost[current][i] ;
      if (newdist < distance[s][i])
      {
/* distance from s to i through current is smaller than
                                                                    distance[s][i] */
          distance[s][i] = newdist;
          precede[s][i] = current;
      } /* end if */
    }
    /* determine the smallest distance */
    if ( distance[s][i] < smalldist )
    {
        smalldist = distance[s][i] ;
        k = i;
    } /* end if */
} /* end for --- if */
current = k;
perm[current] = MEMBER ;
} /* end while */
return;
}

```

```

int umodi(x,y)
  int x,y;
  {
    int k;
    k = ( x < y ) ? x : x - ((int)(x/y)*y) ;
    return(k);
  }

/*-----*/
void departure(i,j)
  int i,j;
  {
    int k;
    struct packet *nptr;
    float sizepkt ;
    if(llst[i][j].f==llst[i][j].r)
      llst[i][j].linkbusy =0;
    else {
      k=llst[i][j].f;
      k= umodi((k+1),5);
      llst[i][j].f = k ;
      nptr = linkqueue[i][j][k];
      linkqueue[i][j][k]=NULL;
      if (nptr-> inorexorc == 1 )
        sizepkt = pktsize ;
      else sizepkt = cpktsize ;
      llst[i][j].linkbusy = 1 ;
      llst[i][j].trf += sizepkt/capacity[i][j];
      llst[i][j].tottrf += sizepkt/capacity[i][j];
      llst[i][j].wttim = mainclock - nptr->inttm;
      llst[i][j].delay += mainclock - nptr->inttm +
                                sizepkt/capacity[i][j];
      nptr->inttm = mainclock + sizepkt/capacity[i][j];
      nptr->nhps++;
      llst[i][j].linkcmt = nptr->inttm;
      insert(nptr);
    }
  }

```

```

        llst[i][j].pktdvd++;
        llst[i][j].intpktdvd++;
        return;
    }
}
/*-----*/
/* The routine on occurrence of an event carries out the departure
functions ,routedecision for incoming packet and its placement in
queue or link */
void arrival(nptr)
    struct packet *nptr;
{
    int i,j,k,flag=0;
    if(nptr->inorexorc == 1 ;; nptr->inorexorc == 2 )
    { i = nptr->intsnd;
      j = nptr->intdnd;
      llst[i][j].linkbusy = 0 ;
      flag = cpktdp(i,j);
      if (flag == 0) departure(i,j);
      if(nptr->dest == nptr->intdnd)
      { if (nptr->inorexorc == 1)
        { mgsdvd++;
          timnk += mainclock - nptr->pkgntm;
          tnhps += nptr->nhps ;
          free ((char *) nptr);
          return;
        }
      }
      else
      { cmgdvd++;
        nptr = cpktrout(nptr) ;
        free((char *)nptr);
        return;
      }
    }
}
/* Checking for max. no. of hops condition . */
if (nptr->nhps > (nonds + 2))
{ imgsrejected ++ ; /**Could be control pkts also **/
  free ((char *) nptr);
}

```

```

return;
}
/**/ Since packet has reached its intermdiate destination ***/
nptr->intsnd = nptr->intdnd ;
} /* end of if */
/**According to routing decision intdnd is updated ***/
nptr->intdnd = destination[nptr->intsnd][nptr->dest] ;

if (nptr->inorexorc == 0 ;; nptr->inorexorc == 1)
{ i = nptr->intsnd;
  j = nptr->intdnd;
  if(llst[i][j].linkbusy == 0)
    {nptr->inorexorc = 1;
     departlf(nptr);
     return;
    }
  k = llst[i][j].r;          /**Circular Queue      **/
  k = umodi((k+1),5); /** Rear Pointer      **/
  if(k== llst[i][j].f)      /**Queue full condition**/
    { if (nptr->inorexorc == 1)
      {mgsrejected ++;
       free((char *)nptr);
       return;
      }
      if (nptr->inorexorc == 0)
      {mgsrejected ++;
       free((char *)nptr);
       return;
      }
    }
  else
    { llst[i][j].r = k ;
      nptr->inorexorc=1;
      linkqueue[i][j][k] = nptr;
      nptr = NUL;
    }
  return;
}

```

```

    }
}
/*-----*/
/* To print output result */
void display()
{
    int i, j;
    float temp[20][20],tot1=0;
    float tot2 = 0;
    void disptb(array);

    for (i=0; i<20; i++)
        for(j=0;j<20;j++)
            temp[i][j] = 0;
    printf(" THE NETWORK CONFIGURATION IS ");
    printf(" Number of nodes : %d ",nonds);
    printf(" NETWORK CONNECTIVITY ( adjacency matrix)");
    disptb(link);
    printf(" LINK CAPACITIES ARE ");
    disptb(capacity);
    printf(" Mean interarrival rate (pkt/sec ) at nodes are :");
    printf(" _____");
    for(i =0;i<nonds;i++)
        printf("%4d ",i);
        printf("");
    for(j=0;j<nonds;j++)
        printf("%4.2f ",lambda[j]);
        printf("_____");
    printf(" Packet Size is : %f bits/sec ",pktsize);
    printf(" Network Simulation time is :%f secs ",simtim);
    printf(" OUTPUT STATISTICS ");
    printf("*****");
    printf(" Total number of message packets generated : %d ",
            (int)msggen);
    printf(" Average number of packets generated : %fpackets/sec.
            ",msggen/simtim);

```

```

printf(" Average number of packets rejected at source: %f
           packets/sec ",mgsrejected/simtim);
printf(" Average number of packets rejected in the network:
           %fpackets/sec ",imgsjrejected/simtim);
printf(" Average number of packets delivered (THROUGHPUT) :
           %f packets/sec "mgdvd/simtim);
printf(" Total %% of packets delivered :%f ",
           mgdvd*100.0/mgs);
printf(" Total %% of packets rejected :%f ",(imgsjrejected +
           mgsrejected)*100/simtim);
printf(" Average number of hops taken per packet :%f ",
           tnhs/(mgdvd+cmgdvd));
printf(" Average time spent by each packet delivered in the
           network : %f ",timnk/mgdvd);
printf(" LINK STATISTICS ");
printf(" Average packets delivered by each link in packets/sec ");
for (i=0; i<nonds; i++)
  for (j=0; j<nonds; j++)
    temp[i][j]= (float)llst[i][j].pktdvd/simtim ;
disptb(temp);
printf("Average waiting time per packet in QUEUE for each link ");
for (i=0; i<nonds; i++)
  for (j=0; j<nonds ; j++)
    temp[i][j] = llst[i][j].wttim/(float)llst[i][j].pktdvd ;
disptb(temp);
printf(" LINK UTILIZATION ");
for(i=0; i<nonds; i++)
  for(j=0; j<nonds; j++)
    if (link[i][j] == 1)
{
temp[i][j] = llst[i][j].tottrf/simtim ;
tot1 += temp[i][j];
tot2++;
}
disptb(temp);

```



```

printf(" Average waiting time per Queue : %f ",tot1/tot2);
printf(" -----");
printf(" COMPREHENSIVE OUTPUT STATISTICS ");
printf(" *****");
printf(" INPUT PARAMETERS : ");
printf(" Number of nodes      : %d ",nonds);
printf(" Simulation Time       : %f secs ",simtim);
printf(" Packetsize is         : %f bits ",pktsize);
printf(" -----");
printf(" OUTPUT STATISTICS :");
printf(" Total no. of message pkts generated      : %d ",
                                           (int)mgsgen);
printf(" Total %% of packets delivered: %f ",
                                           mgsdvd*100.0/mgsgen);
printf(" Total %% of packets rejected: %f ",(imgsrejected +
                                           mgsrejeccted)/mgsgen);
printf(" Average waiting time per Queue      : %f ",tot1/tot2);
printf(" Average number of hops taken / packet %f ",
                                           tnhps/(mgsdvd+cmgdvd));
printf(" Average no.of pkts dvd/sec (THROUGHPUT): %f ",
                                           mgsdvd/simtim);
printf(" -----");

return;
}
/*-----*/
/* For displaying in table format */
void disptb(array)
float array[20][20] ;
{
int k,l;
printf(" _____");
printf(" DESTINATIONS ");
printf(" ");

```

```

    for ( l=0; l<nonds; l++)
        printf(" %d",l);
    for ( k = 0 ; k < nonds ; k++)
        {
    printf(" Source %2d ->",k);
    for ( l = 0 ; l<nonds ; l++)
        if( link[k][l] == 1 )
            printf("%6.2f ",array[k][l]);
        else printf(" ");
        }
    }
printf(" _____");
return;
}
/*-----*/
/* Main function */
main()
{
    int flag = 1 ;
    int i,j,rd,flag1=0;
    long l;
    struct packet *nptr;
    void update();
    void clear();
    updtim=0.2;
    cpktsize=176.0;
    while ( flag == 1 )
        {
            initialise();
            readinput();
            flag = connectedornot();
        }
    genevlst();
    for( i= 0; i<nonds ;i++)
    for( j = 0;j<nonds;j++)
        if ( link[i][j] == 1 )
            {

```

```

cost[i][j] = 1;
delta[i][j] = 0.064;
}
for(i=0 ; i < nonds;i++)
shpdijk(i);
destb();
next : nptr = getevent();
clear();
printf("\n\n\n\n\n\n\n");
printf(" CLOCK = %f ",mainclock);
printf("type is %d " ,nptr->inorexorc);
if(nptr == NUL) goto terminate;
else {
mainclock = nptr->inttm;
if(mainclock >= simtim ) goto terminate ;
else{
switch(nptr->inorexorc){
case 5 :
nptr->inttm += updtim;
insert(nptr);
destb();
break;
case 4 : nptr->inttm += cpgint;
insert(nptr);
if (flag1 < 2)
{for( i =0 ; i<nonds ; i++)
for( j =0 ; j <nonds ; j++)
if ( link[i][j] == 1 )
{
ncost[i][j] = llst[i][j].trf / cpgint ;
llst[i][j].trf = 0 ;
llst[i][j].delay = 0.0;
llst[i][j].intpktdvd = 0.0 ;
}
}
for ( i=0 ;i<nonds;i++)
for ( j=0 ;j<nonds;j++)
cost[i][j] = ncost[i][j];

```

```

        for ( i=0 ;i < nonds ; i++)
            shpdijk(i);
        flag1++;
    }
    else { cpkgen();
        reroute();
    }
    break;
default:    arrival(npnr);

    }
    }
    goto next;
}

terminate : display();
printf(" PLEASE ENTER : ");
printf(" TO UPDATE      : 1 ");
printf(" TO RESTART     : 2 ");
printf(" TO EXIT ANY KEY EXCEPT 1 & 2 ");
printf(" Enter : ");
scanf("%d",rd);
switch (rd) {
    case 1 : break;
    case 2 : main();
            break;
    default: break;
}
}

/*-----*/
/* To update the network parameters */
void update()
{
    int i , ch,j ,k,l,flag;
printf ( "Do You want to change SIMULATION PARAMETERS : (1/0)=");
    scanf("%d",&ch);
    if ( ch !=1) exit(1);
        varinit();

```

```

printf(" Do You want to change NETWORK CONFIGURATION 1/0=");
scanf("%d",&ch);
if ( ch == 1 )
{
printf( " Number of nodes is %d ",nonds );
printf(" Do You want to CHANGE it : 1/0 ");
scanf("%d",&ch);
if ( ch == 1 )
{
nonds =0;
while(nonds > 20 || nonds == 0)
{ printf(" Enter the no. of nodes (<20) : ");
scanf("%d",&nonds);
};

goto next1;
}

printf(" Do You want to change the NETWORK TOPOLOGY :Y/N ");
scanf("%d",&ch);
if ( ch == 1 )
{
next1 : for( i = 0 ; i < nonds ; i++)
for( j = 0 ; j< nonds ; j++)
{
link[i][j]. = -1 ;
capacity[i][j] = 0.0 ;
}
for (k=0; k<nonds;k++)
{ for (l=0 ; l < nonds ;l++ )
{if ( k == l)
{
link[k][l] = 0;
capacity[k][l] = 0 ;
}
else if(k>l)
{ link[k][l] = link[l][k];
capacity[k][l] = capacity[l][k];
}
}
}
}
}

```

```

                else {
                    if (link[k][l]== 1 ; ;link[k][l]==0) flag =TRUE ;
                    else flag = FALSE;
                    while(!flag)
{ printf(" Enter 1 if link %d %d is present else 0 :",k,l);
  scanf("%f",&link[k][l]);
  if(link[k][l]==1 ; ; link[k][l]==0) flag =TRUE;
}
while (link[k][l] == 1 && capacity [k][l]<=0)
{
    printf(" Enter the capacity of links in bits/sec (>0) : ");
    scanf( "%f",&capacity[k][l]);
}
}
}
}
for( k = 0 ; k < nonds; k++)
{
printf(" Enter the mean interarrival rate (pkt/sec) lambda at each no
printf(" Enter lambda for node %d : ",k);
scanf("%f",&lambda[k]);
flag = TRUE ;
}
}
}
if ( flag != TRUE )
{
printf(" Mean interarrival rate (pkt/sec ) at nodes are : ");
for(i =0;i<nonds;i++)
printf("%4d ",i);
printf("\n");
for(j=0;j<nonds;j++)
printf("%2.2d ",lambda[i]);
printf(" Do You want to CHANGE it : 1/0 ");
scanf("%d",&ch);
if ( ch == 1 )
{

```

```

for( k = 0 ; k < nonds; k++)
{
printf(" Enter the mean interarrival rate (pkt/sec) lambda at
each node ...");
printf(" Enter lambda for node %d : ",k);
scanf("%f",&lambda[k]);
}
}

printf(" Packet Size is : %f bits/sec ",pktsize);
printf(" Do You want to CHANGE it : 1/0 ");
scanf("%d",&ch);
if ( ch == 1 )
{
printf(" Enter packet size (in bits) : ");
scanf("%f",&pktsize);
}

printf(" Network Simulation time is :%f secs ",simtim);
printf(" Do You want to CHANGE it : 1/0 ");
scanf("%d",&ch);
if ( ch == 1 )
{
/*Reading the simulation period & updating interval */
printf(" Enter the period for which simulation is to be
carried out in secs");
scanf("%f",&simtim);
}

printf(" The UPDATING INTERVAL is = %f ",cpgint);
printf(" Do You want to CHANGE it : 1/0 ");
scanf("%d",&ch);
if ( ch == 1 )
{
printf(" Enter the UPDATING INTERVAL in secs : ");
scanf ("%f",&cpgint);
}

printf(" The routing algorithm being used is ");

```

```
/* Add switch statement to choose routing algorithm.*/  
printf(" Do You want to CHANGE it : 1/0 ");  
    scanf("%d",&ch);  
    if ( ch == 1 )  
    { printf("*****");  
  
/* Add switch statement to choose routing algorithm.*/  
    }  
}  
  
void clear()  
{  
    putchar(0x1b);  
    putchar(0x5b);  
    putchar(0x48);  
    putchar(0x1b);  
    putchar(0x5b);  
    putchar(0x4a);  
}
```