

98/11  
0  
12

# RELIABILITY OF COMPUTER NETWORKS

BY

BARBARA AMINA M. AL-RAMADNA

عميد كلية الدراسات العليا

*SUPERVISOR*

**PROF. JAMIL AYOUB**

SUBMITTED IN PARTIAL FULFILLMENT OF THE  
REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE IN  
ELECTRICAL ENGINEERING

**Faculty of Graduate Studies  
University of Jordan**

AUGUST 1998

THIS THESIS WAS SUCCESSFULLY DEFENDED AND APPROVED  
ON: AUGUST 4, 1998

EXAMINATION COMMITTEE

SIGNATURE

Chairman: PROF. JAMIL AYOUB



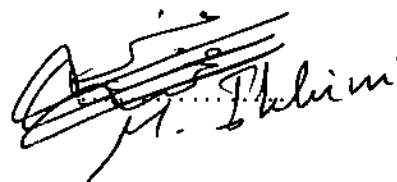
Member: DR. SOUHEIL ODEH



Member: DR. BASSAM KAHHALEH



Member: DR. MOHAMMED IBBINI



## ***ACKNOWLEDGEMENT***

*I AM EXTREMELY GRATEFUL TO MY ADVISOR PROF. JAMIL AYOUB FOR HIS GUIDANCE, HELPFUL COMMENTS, AND SUGGESTIONS, WHICH WERE BEHIND THE SUCCESS OF THIS WORK.*

## Table of Contents

	<b>Page</b>
<b>Committee</b>	ii
<b>Acknowledgement</b>	iii
<b>Table of Contents</b>	iv
<b>Tables List</b>	viii
<b>Figures List</b>	ix
<b>Appendices List</b>	xii
<b>Abstract</b>	xiii
<b>Chapter 1: Introduction</b>	1
1.1 Introduction	1
1.2 Graph Theoretic concepts	3
1.3 Connection Matrix	5
1.4 Routes	6
1.5 Weights	7
1.6 Shortest-route Length Matrix and Successor Node Matrix	9
1.7 Node-Disjoint and Link-Disjoint Routes	9

<b>Chapter 2: Switched Networks</b>	11
2.1 Switching	11
2.2 Circuit Switching	11
2.3 Message Switching	12
2.4 Packet Switching	13
2.5 Routing in Packet-Switched Network	15
<b>Chapter 3: Reliability of Networks</b>	18
3.1 Definitions	18
3.2 Li and Silvesters Approach	20
3.2.1 The Reliability Model	21
3.3 Delay (Kleinrock's Model)	23
3.3.1 Average Time in the System	27
3.3.2 Extension of Kleinrock's Model	30
3.4 Delay as a Performance Measure	32
<b>Chapter4: Average Network Delay Algorithm</b>	36
4.1 Our Model	36
4.2 Routing	38
4.2.1 Shortest Path Routing	39
4.2.2 Dijkstra's Algorithm	40

4.2.3	Multipath Routing	42
4.2.4	Centralized Routing	44
4.2.5	Distributed Routing	46
4.2.6	Main Issues in Routing	47
4.2.7	Routing in Practice	52
4.3	Our Reliability Algorithm	56
4.3.1	New Routing Algorithm	57
4.3.2	The Reliability Algorithm	59
<b>Chapter 5: Results and Conclusions</b>		63
5.1	Introduction	63
5.1.1	Generating the States	64
5.1.2	Shortest Path Calculations	65
5.1.3	Link Flows	66
5.1.4	Average Delay	66
5.1.5	Probability of States	66
5.2	The Most Probable States	68
5.3	Reliability Measure	75
5.3.1	Reliability Measure	79
5.4	Effect of Availability	80
5.5	Calculation Time	84
5.6	Conclusion	88

## Tables List

	<b>Page</b>	
Table 5.1	Probability of states	73
Table 5.2	No and single failure delay	75
Table 5.3	Two failures at a time delay	76
Table 5.4	Probability of states for decreased availability (0.85)	82
Table 5.5	Probability of states for Fig.5.11	88

## Figures List

		<b>Page</b>
<b>Chapter One</b>		
Fig.1.1	Graph	3
Fig.1.2	Undirected graph	4
Fig.1.3	Directed graph	4
Fig1.4	Completely connected graph	7
Fig1.5	Weighted graph	7
 <b>Chapter Two</b>		
Fig.2.1	Packet-switched network	17
 <b>Chapter Three</b>		
Fig.3.1	Single-Server queue	25
Fig.3.2	Communication network	26
 <b>Chapter Four</b>		
Fig.4.1	Interaction of routing and flow control	48
Fig.4.2	Delay-throughput operating curves for good and bad routing	49
Fig.4.3	Routing example	50



Fig.4.4	Routing example	51
Fig.4.5a	ARPANET nodes	53
Fig.4.5b	Routing tables maintained by the nodes in the ARPANET	54
 <b>Chapter Five</b>		
Fig.5.1	Communication network	69
Fig.5.2	Probability distribution function for the no failure and single failure states	70
Fig.5.3	Probability distribution function for two failure states	71
Fig.5.4	Probability distribution function for the no, single, and two failure states	72
Fig.5.5	Probability distribution functions using different loading conditions	74
Fig.5.6	Probability distribution function for decreased availability (0.85)	81
Fig.5.7	Average delay $\gamma = 104\text{Kbps}$	83
Fig.5.8	Average delay $\gamma = 128\text{Kbps}$	84
Fig.5.9	Calculation time	84
Fig.5.10	A 5 node, 7 link network	85
Fig.5.11	Calculation time	86

## Appendices List

		Page
A1 Matlab program	Reliability Algorithm	95
A2 Matlab program for	Calculating the Probability of the States	107
	Generating the Probability Distribution Function	
	Calculating the Average Delay	
A3 Matlab program for	Plotting Results of: Average Time versus Availability	113
	Calculation Time	
	Probability Distribution Function	
A4 Samples of	Input and Output File for Fig. 5.1	116
	Connection Matrix	
	Successor Node Matrix	
	Shortest Route Weight Matrix	
	Link Flows	
	Average Time in the System	

*Abstract***RELIABILITY OF COMPUTER  
NETWORKS***BY***BARBARA AMINA M. AL-RAMADNA***SUPERVISOR***PROF. JAMIL AYOUB**

In this thesis, reliability of packet switched networks subject to random failure is considered.

Classical measures of reliability usually measure a connectivity of some kind, or a probability that two nodes can communicate. Other measures of reliability evaluate a performance measure of the network, assuming no failures. Our model takes all of the above reliability measures into account. It tries to be closer to reality since it depends on the fact that when some of the links of a communication network fail, the whole network does not stop operating, but continues to afford communication at a performance measure which could be evaluated. This performance measure depends on the routing technique used, and on our assumption that the traffic is rerouted in the case of failure. In a real network, associated with each link is an availability, or a probability of correct operation in the case of malfunction. Depending on the link

availabilities, occurrence probabilities of a state of the network could be calculated. Such a state could be a no failure state, i.e., a state where no links fail, or a single failure state; where one link fails, ...etc., up to where all links of the network fail. In our work, the adopted reliability performance measure, is the network average delay, which is calculated for each of the no failure, single failure, and two failure states. The routing model used depends on shortest path routing where shortest refers to the path with the highest availability. Our model uses the most reliable path as a primary path and the second most reliable path, which serves as a backup path in the case of overload conditions in an attempt to minimize the network average delay. After solving a particular network, and calculating its performance measure for the various states considered, the average delay over all states can be calculated. Our algorithm was tested on several different networks. The effect of loading, and availability on network average delay was studied. Also the effect of the size of the network was considered.

The algorithm gives a good and close to reality performance measure, which could be used to measure the performance of operating networks, and in the design of communication networks.

# Chapter One

## Introduction

### 1.1 Introduction

In this thesis, we present a new reliability model for packet switched networks. Classical measures of reliability usually calculate the probability of obtaining service between a pair of nodes, that is, the probability of a pair of nodes being able to communicate (Aggarwal et al.,1981), (Ball,1979).

Other known measures of reliability calculate a specific network performance measure, such as network average delay as in Kleinrock (Kleinrock,1976), which is the time spent by all messages in the network. However, this performance assumes no failures in the network.

Our reliability, or performance measure is intended to be closer to reality as it takes into account the network remaining connected, and being able to provide service in the case of failure. The performance measure which is average network delay according to Kleinrock is evaluated for the network assuming no failures, and for the failure states which are most likely to occur. The reliability measure adopted is the average of this delay over all states considered.

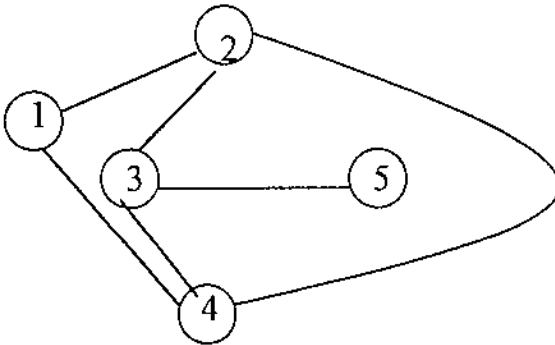
A network consists of links, and nodes. -Chapter one describes graph theoretic concepts which are necessary for our work. Chapter two describes switching.- Associated with each node and link is a probability of it remaining operable. Thus, under the assumption of random failure, a network can take on any of a number of states, with each state having a certain probability of occurrence as will be explained in chapter three.

We have considered all the states judged to be the most probable that the network finds itself in. These states include all states with one link failing at a time and two links at most failing at a time. The performance measure which is the network average delay using Kleinrocks delay, was then evaluated. This is done by taking into account the ability of the network to reroute traffic in the case of failure. A routing policy was adopted which depends on choosing the most reliable path under normal operating conditions, and taking the second most reliable path as a backup in the case of overflow as explained in chapter four.

In chapter five, the results of tests in applying our reliability algorithm to several networks of varying sizes were demonstrated. The effects of loading and link availability on the average delay were studied and documented. The restriction of the state space to those judged as the most probable states was also justified and demonstrated.

## 1.2 Graph Theoretic Concepts

A communication network consists of transmission links that connect locations such as switching centers, concentrators, or terminals. This network is modelled by a graph consisting of a finite set of nodes representing the switching centers which are connected by lines, called links, which represent the transmission links (Cravis,1981), (Bertsekas et al.,1987). Fig. 1.1 is an example of a graph.



**Fig. 1.1 Graph.**

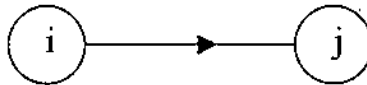
In this graph, we have the set of nodes  $\{1, 2, 3, 4, 5\}$  and the set of node pairs (links)  $\{(1, 2), (2,3), (2,4), (1,4), (3,4), \text{ and } (3,5)\}$ .

If the set of nodes is  $N$  and the set of links is  $L$ , we will refer to the graph as  $G(N, L)$ . Graphs could be directed or undirected. The graph in Fig. 1.1 is undirected, that is each link is arranged for communication in both directions. Fig. 1.2 is another example of an undirected graph representing only one link  $(i, j)$  or  $(j, i)$  with  $i$  and  $j$  being node numbers.



**Fig. 1.2 Undirected graph.**

A directed graph or digraph is a finite nonempty set  $N$  of nodes and a collection  $L$  of ordered pairs of distinct nodes from  $N$ ; where each ordered pair of nodes  $(i, j)$  is arranged for communication from node  $i$  to node  $j$  with an arrow placed on the link to indicate that direction, as shown in Fig.1.3.



**Fig. 1.3 Directed graph.**

A route in an undirected graph from  $i$  to  $j$  is a subset of  $L$  whose members can be arranged in an ordered list  $(i, i_1), (i_1, i_2), \dots, (i_k, j)$  with the following properties:

- 1-  $i$  appears only as one end of the first link, and  $j$  appears only as one end of the last link.



2- If  $k \geq 1$ , every other node  $i_1, i_2, \dots, i_k$  appears exactly twice, as the common end of some link in the list and the link immediately following it.

A graph is connected if, for any distinct nodes  $i$  and  $j$ , there is a route from  $i$  to  $j$ . A disconnected graph has no route between at least one pair of nodes  $i$  and  $j$ .

A component of  $G$  is a maximal connected subgraph  $G'$ , such that no other subgraph that contains  $G'$  is connected.

A link cut-set of  $G$  is a minimal set of links  $L'$  which is a subset of the links  $L$  of a connected graph  $G$ ; such that removing the links in  $L'$  results in a graph with two or more components, and no subset of  $L'$  other than  $L'$  itself has this property.

A node cut-set of a connected graph  $G$ , is a minimal set of nodes  $N'$  which is a subset of the nodes  $N$ , such that, removing the nodes in  $N'$  results in a graph with two or more components, and no subset of  $N'$  other than  $N'$  itself has this property.

### 1.3 Connection Matrix

If a graph  $G(N, L)$  contains  $n$  nodes, it can be described by the  $n$ -by- $n$  connection matrix  $A$  whose element in the  $i$ th row and  $j$ th column  $a_{ij}=1$  if there is a link  $(i,j)$  or  $a_{ij}=0$  if there is no such link. For undirected

graphs  $A$  is symmetric, that is,  $a_{ij}=a_{ji}$ . For the graph of Fig. 1.1. The connection matrix is shown below:

$$A = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

## 1.4 Routes

A problem in communication networks is to find one or more routes between a specified pair of nodes, or between all pairs of nodes, subject to certain constraints, such as cost, delay, network congestion, or the user requested quality of service. Disregarding constraints the number of routes depends on the number of nodes. For a completely connected graph with  $n$  nodes, the number of routes for any node pair approaches  $e^{(n-2)!}$ , where  $e=2.718\dots$ (Cravis,1981). Completely connected means that there exists a link between each pair of nodes in the graph. Fig. 1.4 is an example of such a graph.

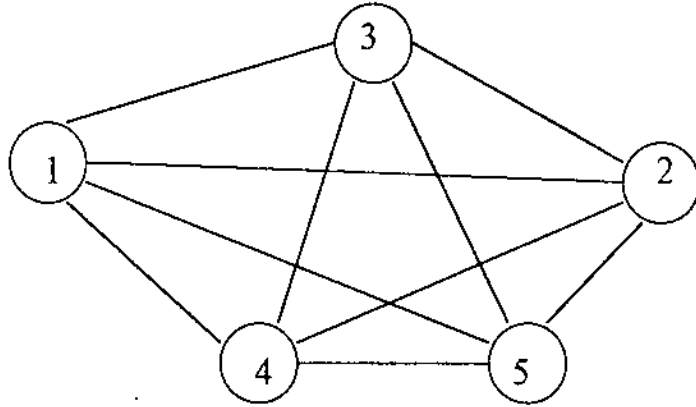


Fig. 1.4 Completely connected graph.

### 1.5 Weights

It is common in graphs to assign to each link a real number called its weight. For example, the weight may be the length of the link, its cost, its transmission capacity, transit delay, or any other useful quantity. Weights are shown as numbers next to the links, as in Fig. 1.5. For example link (2,4), has weight 3. In an undirected graph, link  $(i, j)$  and link  $(j, i)$ , are the same. Also, the weight of link  $(i, j)$  and that of link  $(j, i)$  is the same. Thus if  $w_{ij}$  is the weight of link  $(i, j)$ , one has  $w_{ij}=w_{ji}$ . Also  $w_{ii}=0$ .

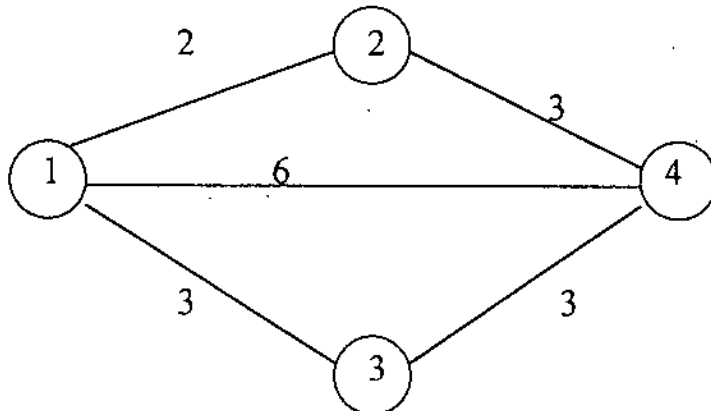


Fig. 1.5 Weighted graph.

The weight of a route is the sum of the weights of its links. For example, in Fig. 1.5, the route (1, 2), (2, 4) from node 1 to node 4 has weight 5, and the route (1, 4) has weight 6.

We will assume that a link weight is a nonnegative quantity. A simple special case is where the weight of each link is 1; the weight of a route is then the number of links it contains.

In our present work, we will deal with a connected graph  $G(N, L)$  having a set of nodes,  $N$ , and a set of links,  $L$ . Each link  $(i, j)$  has a weight  $w_{ij}$ , with  $w_{ij}=w_{ji} \geq 0$ , and  $w_{ii}=0$ ; if there is no link  $(i, j)$ ,  $w_{ij}=\infty$ . "Length" usually is a synonym for "weight", and "shortest" is a synonym for "minimum weight". In describing a route, the triangle inequality,  $d_{ij}+d_{jk} \geq d_{ik}$  for distinct  $i, j, k$  nodes, does not necessarily hold.

The weight measure used in this thesis means that, for example, if for each link,  $p_{ij}$  is the probability that the link remains operable in the face of possible destruction or malfunction, and when we define a weight  $w_{ij} = -\log p_{ij}$ , then a minimum-weight route is one with the highest probability of being operable, assuming that this probability for a route is the product of the link probabilities. These concepts will be considered more formally in the sequel.

## 1.6 Shortest-route Weight Matrix and Successor Node Matrix

For a graph  $G(N,L)$ , the shortest route length matrix  $S$  is a symmetric  $n$ -by- $n$  matrix whose  $i, j$ th element  $s_{ij}$  equals the length of the shortest route between node  $i$  and  $j$ . The successor node matrix  $R$ , is a  $n$ -by- $n$  matrix whose  $i, j$ th element  $r_{ij}$  equals the successor node of  $i$  in the shortest route between  $i$  and  $j$ . In the definition of a route between  $i$  and  $j$ , in Section 1.2, the successor node of  $i$  is the node following  $i$  in the list of links,  $i_1, i_2, \dots, i_k$ . It is possible for  $i_j$  to be equal to  $j$ .

For the example of Fig. 1.5, the shortest route length matrix  $S$  and the successor node matrix  $R$  are given below:

498731

$$S = \begin{bmatrix} 0 & 2 & 3 & 5 \\ 2 & 0 & 5 & 3 \\ 3 & 5 & 0 & 3 \\ 5 & 3 & 3 & 0 \end{bmatrix}$$

$$R = \begin{bmatrix} 0 & 2 & 3 & 2 \\ 1 & 0 & 1 & 4 \\ 1 & 1 & 0 & 4 \\ 2 & 2 & 3 & 0 \end{bmatrix}$$

## 1.7 Node-Disjoint and Link-Disjoint Routes

If each route, in some set of routes between nodes  $i$  and  $j$ , passes through one or more of the same intermediate nodes, service on all the routes is interrupted if messages cannot get through any of the common

nodes, by reason of damage or unavailability. Similarly, if several routes contain one or more links in common, service on all the routes is interrupted if any of the common links fails. This leads to what is called node-disjoint routes (Cravis,1981), that is, those routes that pass through no common nodes except  $i$  and  $j$ , and link-disjoint routes where no links are shared among the routes.

In some analysis methods, it is generally it is not only required that routes have a disjointness property, but also that they meet some additional constraints. For example, if we need just two routes between nodes  $i$  and  $j$ , we usually use (1) the shortest route, and (2) the shortest route that is node-disjoint from (1).

## Chapter Two

### Switched Networks

#### 2.1 Switching

Communication in computer networks means that one network user should be able to exchange information with many other users. This is implemented by transporting bits over the network. The objective of the bit transport is connectivity. Thus, it should be possible to route the bits of one user to all other users. The property of being able to vary the path followed by the bits is called switching. The objective of switching is to share hardware resources while achieving connectivity (Walrand,1991).

#### 2.2 Circuit Switching

In circuit switching the actual physical electrical path or circuit between the source and destination nodes must be established before data can be transmitted. After the establishment of the connection, the use of the circuit is exclusive and continuous for the duration of the information exchange. When the information has been transferred, the circuit is disconnected and the physical links between the nodes are ready for use by other connection.

Circuit switching is mostly used in telephony. For information-transmission, this type of switching is expensive, and inefficient, since the total capacity is used during the transmission. Examples include bit-stream and character-by-character terminal communication. Once a connection is established, delivery is guaranteed. Packet sizes are small, and communication takes place in real time.

### 2.3 Message Switching

In this switching method, no dedicated path is established in advance between sender and receiver. Instead, the message is forwarded technology. When a node has a message to send, it is placed in the first switching office and then forwarded to the next office. The message hops from node to node. At each node, the message received, is inspected for errors, and is stored in a buffer until storage until a link to the next node is available. The message is then forwarded to the next node.

In message switching, the message is stored in a buffer. For example, in an electronic mail system, a message is a document sent from one user to another. In a network, a message between two or more users, a message is a document sent from one participant to another. A message in a network system could be a single figure, image or image.



messages as one complete unit is generally not done because large storage capacities must be available at each node to buffer these long blocks. As a result, a single message block can tie up an internodal line for many minutes. Therefore, long messages are normally broken up into shorter bit strings called packets. These packets are then sent through the network as individual units and reassembled into complete messages at the destination station. This leads to what is called packet-switching. On the other hand in message-switching, no real end-to-end path is created. This makes good use of the communication channels since no resources need to be reserved until the channels are available.

## **2.4 Packet Switching**

Packet-switching is very similar to message-switching. A packet-switching system accepts packets from an information source, stores them in buffer memory, and then forwards them to the next packet-switch where the same store-and-forward operation occurs. The only difference is that the maximum length of any packet is generally very short (1000 to 5000 bits) and therefore does not encounter a long transmission delay as it propagates through the store-and-forward network (Keiser,1989).

The two techniques for sending packets from node to node in a packet-switching system are the datagram method and the virtual-circuit method. In a datagram transport, the successive packets are sent

independently of one another. It is therefore possible for different packets from the same file to follow different paths in the network and for those packets to arrive in an order that differs from that in which they were transmitted. Each packet is supplied with a sequence number, in order to reassemble them at the receiver side. In a virtual-circuit transport, the different packets that are part of the same information transfer are sent along the same path; the packets follow one another as if they were using a dedicated circuit even though they may be interleaved with other packet streams (Walrand,1991).

Datagram packet-switching is similar to message-switching, in that there is also no need for a call set-up procedure. However, datagram packet-switching is significantly faster than message-switching which makes it suitable for interactive traffic. This is because an upper limit is placed on packet size so that no user can monopolize any transmission line for more than a few tens of microseconds. Another reason is that each node along the route can forward any packet of a multipacket message as soon as it arrives; it does not have to wait for the next packet. This reduces delay and improves throughput. Datagram packet-switching is efficient with short messages and for networks where user flexibility is desired (Keiser,1989).

Virtual-circuit packet-switching is similar to circuit-switching in that a circuit route must first be established. This is initiated by a call-request

packet. The virtual circuit is established once a call-accept packet from the destination node arrives at the source node. When established, the message is transmitted in packets (Keiser,1989).

Every packet experiences some processing delay at each node in the path because of the queuing process. This delay is variable and becomes longer when the network gets more heavily used. Virtual-circuit packet-switching is used in interactive traffic. It is efficient for long exchanges of messages and for relieving stations of processing burdens (Keiser,1989).

Packet-switching is commonly used for terminal-to-computer and computer-to-computer communications. Here the information exchange is viewed as a process wherein bursts of information occur with a high peak-to-average rate. Examples of this are point-of-sale system, inquiry-response systems, remote time sharing services, and electronic message systems (Keiser,1989).

## **2.5 Routing in Packet-Switched Networks**

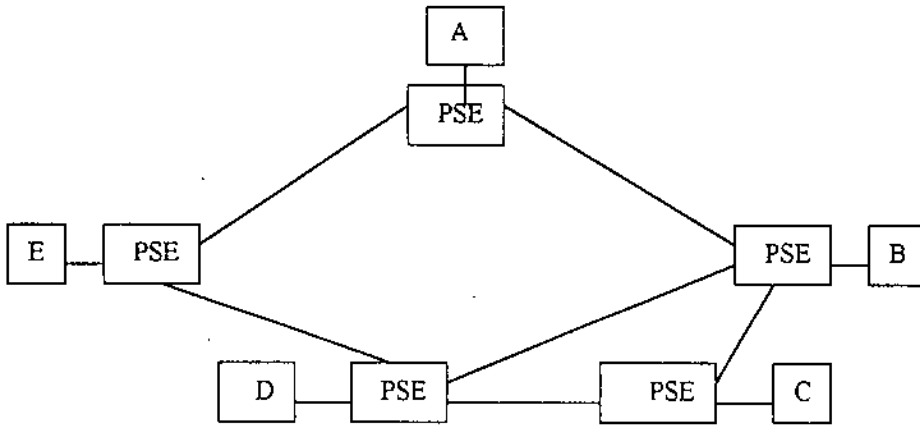
In a packet-switched network all data to be transmitted is first assembled into one or more message units, called packets, by the source node. These packets include both the source and destination node network addresses. They are then passed bit serially by the source node to its local packet switching exchange. On receipt of each packet, the exchange first stores the packet and then inspects the destination address it contains. Each

packet switching exchange contains a routing directory specifying the outgoing links to be used for each network address. The packet switching exchange forwards the packet on the appropriate link at the maximum available bit rate. Similarly, as each packet is received and stored at each intermediate node along the route, it is forwarded on the appropriate link interspersed with other packets being forwarded on that link. At the destination packet-switching exchange, determined by the destination address within the packet, the packet is finally passed to the destination node (Halsall,1995).

A number of packets may arrive simultaneously at a packet-switching exchange on different incoming links and each may require forwarding on the same outgoing link. If a number of particularly long packets are waiting to be transmitted on the same link, other packets may experience unpredictable long delays. To prevent this and to ensure that the network has a reliably fast transit time, a maximum length is allowed for each packet. Therefore, in packet-switched networks a message submitted to the node, is first divided by the source node into a number of smaller packet units before transmission. These smaller units are reassembled into a single message by the destination node (Halsall,1995).

Another feature of packet-switched networks, is the application of sophisticated error and flow control procedures on each link by the network packet-switching exchanges. Therefore, the class of service provided by

packet-switching is much higher than that provided by circuit-switched networks. A packet switched network is shown in Fig. 2.1.



PSE: Packet-switching exchange

**Fig. 2.1 Packet-switched network.**

## Chapter Three

### Reliability of Networks

#### 3.1 Definitions

Reliability is a connectivity measure that is the ability of a network to continue to afford communications routes between some nodes when other nodes or links have failed. There are two kinds of reliability measures: Deterministic and Probabilistic. Deterministic measures depend on the structure of the network whereas probabilistic measures depend also on the probabilities of failure or availability of nodes and links.

One of the deterministic measures of reliability is cohesion. If  $i$  and  $j$  are distinct nodes of a connected graph  $G(N, L)$ , then an  $i, j$  link cut-set of  $G$  is one whose removal results in a graph with two components, one containing  $i$  and the other containing  $j$ . If  $v_{ij}$  is the minimum number of links that must be removed to break all routes between  $i$  and  $j$ , then the cohesion of  $G$  is the minimum of  $v_{ij}$  for all node pairs, and is denoted by  $v$ . In other words it is the minimum number of links that must be removed from  $G$  to break all routes between at least one pair of nodes.

Another measure is connectivity. An  $i, j$  node cut-set of  $G$  is one whose removal results in a graph in which  $i$  and  $j$  are in separate components. If  $w_{ij}$  is the minimum number of nodes in an  $i, j$  cut-set that

must be removed from  $G$  to break all routes between  $i$  and  $j$  (it is also the maximum number of node-disjoint routes between  $i$  and  $j$ ), then the connectivity of  $G$ , denoted by  $w$ , is the minimum of  $w_{ij}$  for all node pairs. It is the minimum number of nodes that must be removed from  $G$  to break all routes between at least one pair of nodes (Cravis,1981).

An alternative approach to reliability is to assign failure probabilities to all network elements (nodes and links), and evaluate the probability that a given pair of nodes becomes disconnected. The difficulties involved in this approach when trying to obtain exact solutions, is that suppose that there are  $n$  failure-prone elements in the network. Then, the number of all distinct combinations, or states, of surviving elements that need to be considered is  $2^n$  (Bertsekas et al.,1987). Let  $S_k$  denote the  $k$ th combination of the network elements, and  $P_k$  denote the probability of its occurrence. Then the probability that the network remains connected is  $P_c = \sum_{S_k \in C} P_k$ ; where  $C$  is the set of combinations of surviving elements for which the network remains connected. Thus, evaluation of  $P_c$  involves some form of implicit or explicit enumeration of the set  $C$ . This is, in general, a very time consuming procedure. However, shortcuts and approximations have been found making the evaluation of the survival probability  $P_c$  feasible for some realistic networks (Bertsekas et al.,1987).

An example is the problem of calculating the terminal reliability for a particular node pair  $s, t$ ; that is, the probability  $Q_{st}$  that at least one operable route exists from  $s$  to  $t$  (Cravis,1981).

An alternative approach calculates bounds on reachability, or connectedness. Another approach evaluates a lower bound for  $P_c$  by ordering the possible combinations of surviving elements according to their likelihood, and then adding  $P_k$  over a subset of most likely combinations from  $C$  (Li et al.,1984). A similar approach using the complement of  $C$  gives an upper bound to  $P_c$  (Bertsekas et al.,1987).

Li and Silvester (Li et al.,1984) presented a solution technique which is not doomed by the state space explosion. Instead of enumerating all possible fail states, only the most probable states are considered. Since the network operates in these states most of the time, one can get upper and lower bounds, and hence, a good approximation of the network performance without having to analyse all possible states. This is explained in section 3.2 below.

### **3.2 Li and Silvesters Approach (Li et al.,1984)**

In this approach rather than enumerating all possible fail states, a state of a network is the condition of the network at any moment in time, where it contains both faulty and operable components. A network of  $L$  fault prone links has  $2^L$  states.- Only the most probable states, say the  $m$



most probable states in a network with  $n$  failure-prone components sufficient to cover a high proportion of the total state space are considered. Bounds can then be given on the performance measure or a probabilistic statement can be made about it.

### 3.2.1 The Reliability Model

For a network with  $n$  unreliable components, the following assumptions can be made.

- 1- Component  $i$ , for  $i = 1, \dots, n$  is in the operational mode with probability  $p_i$ , and in the failed mode with probability  $q_i = 1 - p_i$ .
- 2-  $\frac{1}{2} \leq p_i \leq 1, i = 1, \dots, n$ .

The states of the system are denoted by  $S_k, k = 1, \dots, 2^n$

The occurrence probability of  $S_k$  is given by

$$P(S_k) = \prod_{i=1}^n p_i (q_i / p_i)^{T_i(S_k)} = \left( \prod_{i=1}^n p_i \right) \left( \prod_{j=1}^n R_j^{T_j(S_k)} \right) \quad (\text{Eq. 3.1})$$

where,

$$T_i(S_k) = \begin{cases} 0 & \text{if component } i \text{ is operational in state } S_k \\ 1 & \text{otherwise} \end{cases}$$

$$R_i = q_i / p_i \quad i=1, 2, \dots, n \quad R_1 \geq R_2 \geq \dots \geq R_n$$

The most probable state,  $S_0$ , corresponds to no failures, and the next most probable state is the one in which the mode of exactly one component has changed (from operational to failed). This component is the one with the largest  $R_i$ , i.e., component 1. The states ( $S_k$ ) are generated in order, according to their probability of occurrence. This approach allows bounds on any performance measure for any failure probabilities.

Let  $X$  be the performance measure of interest, and suppose that it takes on the value  $X(S_k)$  when the network is in the state  $S_k$ . This performance measure could be a connectivity measure, or any other performance measure like delay. Then another assumption is

$$X(S_2^n) \leq X(S_k) \leq X(S_0) \quad k = 1, \dots, 2^n - 1, \quad (\text{Eq. 3.2})$$

where  $S_2^n$  corresponds to the state when all components have failed, i.e., the network performance is best when no components have failed and worst when all components have failed.

$E[X]$  is the expected value of the performance measure, where we are averaging over all possible states, is of interest. Upper and lower bounds on  $E[X]$  are defined. When the  $m$  most probable states are considered,  $X_U(m)$ , and  $X_L(m)$  are defined as the upper and lower bound on the performance measure. The performance measure of interest is evaluated by

an appropriate algorithm for  $S_0$  up to  $S_m$ . The lower and upper bounds are thus,

$$X_L(m) = \sum_{k=1}^m P(S_k)X(S_k) + \left(1 - \sum_{k=1}^m P(S_k)\right)X(S_2^n) \quad (\text{Eq. 3.4})$$

$$X_U(m) = \sum_{k=1}^m P(S_k)X(S_k) + \left(1 - \sum_{k=1}^m P(S_k)\right)X(S_0)$$

This approach works for any failure probabilities, and it allows bounds to be obtained on any performance measure. Those bounds rapidly converge as additional states are considered.

### 3.3 Delay (Kleinrock's Model)

The performance measure we will consider in this section is delay. Kleinrock (Kleinrock,1976) determined the average message delay for all messages in a communication network. Before considering delay, the following assumptions and definitions must be made.

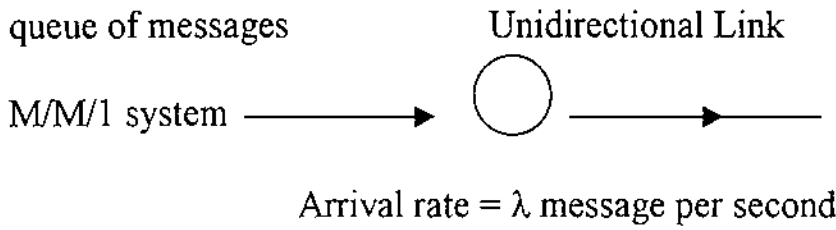
Assumptions: (The Single-Server Queue). Our starting point is to consider a network model with  $M$ -channels,  $N$ -nodes to be used for message switching communication. The  $M$  communication channels are assumed to be noiseless, perfectly reliable, and to have a capacity denoted

by  $C_i$  (bits per second) for the  $i$ th channel. The  $N$  nodes refer to the message -(or packet)- switching centers (IMPs: Interface message processors), which are generally computers to route messages through the network. It is assumed that the nodal processing times are constant with value  $K$  (assumed negligible). In addition to channel queuing and transmission delays, in high-speed networks spanning large geographical regions, it may be important to include the propagation time  $T_p$ .

Traffic entering the network from external sources (HOSTS) which are computers, terminals, or local area networks, forms a Poisson process with a mean  $\gamma_{ij}$  (messages per second) for those messages originating at node  $i$  and destined for node  $j$ . The total external traffic entering or leaving the network is defined by

$$\gamma = \sum_{i=1}^N \sum_{j=1}^N \gamma_{ij} \quad (\text{Eq. 3.5})$$

Message are assumed to have lengths that are drawn independently from an exponential distribution with mean  $1/\mu$  (bits). And nodes are assumed to have unlimited storage capacity. For the analytical results messages are assumed to be directed through the network according to a fixed routing procedure. Each channel in the network is considered to be a separate server. One node is isolated as shown in Fig.3.1



M: Poisson nature of the distribution of message arrivals.

M: Negative exponential distribution of message lengths.

1 : The number of servers.

**Fig. 3.1 Single-Server Queue.**

Consider messages that are queued for transmission outward from this node on one unidirectional link, which is the “server”. Assume that the arrivals of messages at the node have a Poisson distribution. Thus, the probability that exactly  $j$  messages will arrive in an interval of  $t$  seconds is

$$P_j(t) = (\lambda t)^j \exp(-\lambda t) / j!, j = 0, 1, 2, \dots \quad (\text{Eq. 3.6})$$

The interarrival times between successive message arrivals have a negative exponential distribution with mean  $1/\lambda$  seconds. The average rate of message arrivals, the throughput, is  $\lambda$  messages per second. The service time, for a message  $b$  bits in length is  $b/C_i$  seconds.

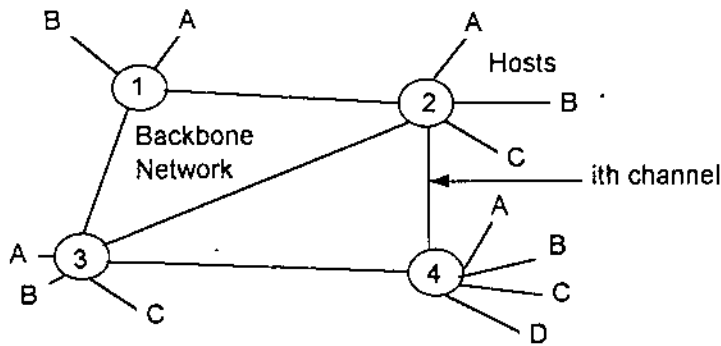
Another assumption is that the stored messages are served, that is, transmitted, in order of their arrival; this is the first-in, first-out (FIFO)

queue discipline. Also, nodes are assumed to have an arbitrarily large amount of storage for messages in queue.  $\lambda$  and  $\mu$  are assumed to be constant, not dependent on the time. Furthermore, we assume that the system is in statistical equilibrium, which has been attained by allowing messages to queue up and to be transmitted on the link for a time that is at least several times the average message service time  $1/C\mu$  seconds.

Since each channel in the network is considered to be a separate server, let  $\lambda_i$  be the average number of messages per second which travel over the  $i$ th channel. The total traffic within the network is defined as

$$\lambda = \sum_{i=1}^M \lambda_i .$$

A model of a communication network is shown in Fig.3.2.



1, 2, 3, 4 : Switching centers (IMP's), Nodes

$\gamma_{ij}$  : Traffic (messages) entering the network originating at node  $i$  and destined for node  $j$ .

$\lambda_i$  : Messages per second which travel over the  $i$ th channel.

A, B, C, D: Hosts

**Fig. 3.2 Communication network.**

### 3.3.1 Average Time in the System

According to Kleinrock, the message delay is defined as the total time that a message spends in the network. The average message delay is  $T = E[\text{message delay}]$  (Kleinrock, 1976). This is our basic performance measure. Define the quantity

$Z_{ij} = E [\text{message delay for a message whose source is } i \text{ and whose destination is } j]$

$$\text{This means } T = \sum_{i=1}^N \sum_{j=1}^N \frac{\gamma_{ij}}{\gamma} Z_{ij} \quad \text{Eq. 3.7}$$

where  $T$  is the average time in the system for all messages which travel through the network, since the fraction  $\gamma_{ij}/\gamma$  of the total entering message traffic will suffer the delay  $Z_{ij}$  on the average.

Assuming a fixed routing procedure, the average rate of message flow,  $\lambda_i$ , on the  $i$ th channel is equal to the sum of the average message flow rates of all paths that traverse this channel,

$$\lambda_i = \sum_j \sum_{i,j: C_i \in \pi_{ij}} \gamma_{ij} \quad \text{Eq. 3.8}$$

where  $\pi_{ij}$  is the path taken by a message originating at node  $i$  and destined for node  $j$ , i.e., the  $i$ th channel of capacity  $C_i$  is included in the path  $\pi_{ij}$  if the message passes through that channel.

This means that  $Z_{ij}$  is the sum of the average delays encountered by a message in using the various channels along the path  $\pi_{ij}$ . The individual channel delays, that is, the average time in the  $i$ th channel is

$$T_i = E [\text{Time spent waiting for and using the } i\text{th channel}]$$

Thus,

$$Z_{ij} = \sum_{i: C_i \in \pi_{ij}} T_i \quad \text{Eq. 3.9}$$

The average delay in the network is

$$T = \sum_{i=1}^N \sum_{j=1}^N \frac{\gamma_{ij}}{\gamma} \sum_{i: C_i \in \pi_{ij}} T_i \quad \text{Eq. 3.10}$$

Exchanging the order of summations yields



$$T = \sum_{i=1}^M \frac{T_i}{\gamma} \sum_i \sum_j \gamma_{ij} \quad \text{Eq. 3.11}$$

$i, j: C_i \in \pi_{ij}$

Using Eq. 3.8 yields

$$T = \sum_{i=1}^M \frac{\lambda_i T_i}{\gamma} \quad \text{Eq. 3.12}$$

This means that the average message delay is composed of its single channel components the delays,  $T_i, \dots$ . Therefore, our analysis reduces to the calculation of  $T_i$ .

Considering the  $i$ th channel as an  $M/M/1$  system with poisson arrivals at a rate  $\lambda_i$  and exponential service times of mean  $1/\mu C_i$  sec, using Little's result the average time in the  $i$ th channel is (Kleinrock,1976)

$$T_i = \frac{1}{\mu C_i - \lambda_i} \quad \text{Eq. 3.13}$$

Using Eq. 3.12 we obtain

$$T = \sum_{i=1}^M \frac{\lambda_i}{\gamma} \left[ \frac{1}{\mu C_i - \lambda_i} \right] \quad \text{Eq. 3.14}$$

In generating this result we have assumed the nodal processing time  $K=0$ , and the propagation delay  $T_p=0$ .

### 3.3.2 Extension of Kleinrock's Model

Kleinrock's model of a message-switched network, can be extended to apply to packet-switched networks (Cravis,1981).

#### The single-packet case

For a message that consists of a single packet, the average time in the system has two components: the service time, which is the sum of the times it takes to transmit the packet on each link it traverses in the backbone network, and the delay time, which is the sum of the delays that the packet experiences at each node waiting for a link to become available. Retaining the assumptions of the single-server queue, and assuming Poisson arrivals of packets, and a negative exponential distribution of packet lengths, the packet transmission time on link  $i$  is  $1/\mu C_i$  seconds. The delay time at a node is  $\tau_i = (\lambda_i/\mu C_i)/(\mu C_i - \lambda_i)$ , where now  $1/\mu$  is the average packet length in bits,  $C_i$  is the capacity of link  $i$  in bits per second.  $\lambda_i$  is the arrival rate of packets at link  $i$  in packets per second.

In a packet-switched network there are a significant number of control or overhead packets transmitted such as for acknowledgements of correct receipt of data packets, that are shorter than data packets. To take this into account let  $1/\mu'$  be the average packets length, both control and data. Also let  $\lambda_i'$  be the total arrival rate of both control and data packets, at link  $i$ . Assume that the single-server queue assumptions apply to the mixed

stream of control and data packets, then the average delay for all packets traversing link  $i$  is

$$W'_i = \frac{(\lambda'_i / \mu' C_i)}{(\mu' C_i - \lambda'_i)} \quad (\text{Eq. 3.15})$$

We must also include the link propagation time, and the nodal processing time, which are both greater relative to the service time in packet switching. Assume that the link propagation time is  $P_{ti}$ , seconds for link  $i$  and the nodal processing time is  $K$  seconds per node. Therefore the average time in the system for a single-packet message is:

$$T_{sp} = K + \sum_i^M \left( \frac{\lambda_i}{\gamma} \right) \left[ \left( \frac{1}{\mu C_i} \right) + W'_i + T_p + K \right] \quad (\text{Eq. 3.16})$$

where  $\gamma$  is total arrival rate of data packets into the backbone network.  $M$  is the number of links in the network. The extra delay outside the summation accounts for each packet's traversal of a number of nodes that is one greater than the number of links it traverses.

The above equation is quite complex. For the ARPANET for example, which is a packet-switched network in the United States, developed by the US Dept. of Defence Advanced Research Projects Agency (DARPA), as an early internet, used to interconnect the computer networks associated with a small number of research and university sites. Since that time, the internet has grown steadily. There are now a large number of workstations, and several thousands networks/subnets.

ARPANET is now linked to other networks. The combined internet, which is funded by a number of different agencies, is simply known as the Internet (Halsall,1995).

We can neglect the  $w_i'$  terms. We can also assume that the link capacities are all the same,  $C_i=C$ . Then the above equation reduces to (Cravis,1981):

$$T_{sp} \approx (n+1)K + \left(\frac{n}{\mu C}\right) + \sum_i \left(\frac{\lambda_i}{\gamma}\right) T_p \quad (\text{Eq. 3.17})$$

$n$  may have a value in the range of 1 to 10: For the ARPANET  $n=3.31$ .  $K$  the nodal processing time depends on the speed of the nodal processor.  $T_p$  the propagation time depends on the properties of the links and their lengths.

### 3.4 Delay as a Performance Measure

The delay for a network with fixed topology (no failures) as shown above is

$$T = \frac{1}{\gamma} \sum_i \frac{\lambda_i}{\mu C_i - \lambda_i} \quad (\text{Eq. 3.18})$$

In deriving Eq. 3.18, we have assumed zero processing and propagation delays, since both the processing time and the propagation time depend on both the characteristics of the switching centers, and on the

characteristics of the transmission links used. Thus, the  $K$  and  $T_p$  terms in Eq. 3.16 differ depending on the particular network used in performing the tests.

By setting  $T_p = 0$ , and  $K=0$ , Eq. 3.16 reduces to

$$T_{sp} = \sum_i^M \left( \frac{\lambda_i}{\gamma} \right) \left[ \left( \frac{1}{\mu C_i} \right) + W'_i \right] \quad (\text{Eq. 3.19})$$

the same assumptions were made in (Gavish et al.,1992), (Thaker et al,1986).

Another assumption made to arrive at Eq. 3.18 is considering  $\mu$  equal the average packet length (Kleinrock,1976), that is, the average length of both the acknowledgement and information packets, i.e.,  $\mu = \mu'$ . Thus, Eq. 3.19 is reduced to Eq. 3.18.

In our analysis we have considered single-packet messages, that are messages which require no more than one packet, this was done due to the close agreement between theory and practice; since most of the traffic in a real network is composed of single-packet messages (Kleinrock,1976).

This formula can then be used to determine the delay in each of the fail states. This delay is the performance measure we will consider.

The following bounds can be defined on  $\bar{T}$  (the expected value of the average delay over all states):

$$T_L(m) = \sum_{k=1}^m P(S_k)T(S_k) + \left(1 - \sum_{k=1}^m P(S_k)\right)T(S_0) \quad (\text{Eq. 3.20})$$

$$T_u(m) = \sum_{k=1}^m P(S_k)T(S_k) + \left(1 - \sum_{k=1}^m P(S_k)\right)T(S_{2^n})$$

where  $m$  is the number of states considered.  $P$ ,  $T$ , and  $S_k$  as previously defined in section 3.2.1.

The “failed mode” in this case corresponds either to the total loss of the link, or reduction in capacity due to noise or jamming. In either case,  $\bar{T}$  will be infinite for values of  $\gamma$  greater than the throughput that can be supported by the network, and thus does not give much information. In particular, if failure corresponds to total loss of use of the component, then  $\bar{T}$  is always infinite.

However if the network is at least connected in all states, the following estimator  $\tilde{T}_m$  for the network average delay over all states can be used (Li et al.,1984):

$$\tilde{T}_m = \frac{\sum_{k=1}^m P(S_k)T(S_k)}{\sum_{k=1}^m P(S_k)} \quad (\text{Eq. 3.21})$$

where  $T(S_k)$  is the delay found for the state  $S_k$ . This is only useful as a performance measure if the state changes are occurring rapidly.

Another approach is to study the distribution of network average delay. Define the probability distribution function for the network average delay to be  $F_T(t)$ . Define (Li et al.,1984)

$$\begin{aligned} \tilde{F}_m(t) &= \sum_{\substack{S_k : T(S_k) \leq t \\ k \leq m}} P(S_k) \end{aligned} \quad (\text{Eq. 3.22})$$

where  $\tilde{F}_m(t)$  is the probability distribution function considering only  $m$  states.

Then

$$F_T(t) \geq \tilde{F}_m(t),$$

i.e.,  $\tilde{F}_m(t)$  is a lower bound on the distribution function for the network average delay.

In our approach, the performance measure of interest, which is network average delay  $T$  will be averaged over all states considered using Eq. 3.21 in order to get a reliability measure of interest.

Using Eq. 3.22 , the distribution of the network average delay will be studied to see if the number of states we considered is adequate to cover a large portion of the state space.

## Chapter Four

### Average Network Delay Algorithm

#### 4.1 Our Model

The network contains switching centers and communication channels. This can be formalized into a graph  $G(N, L)$  where  $N$ , the set of nodes, is the collection of switching stations, and  $L$ , the set of links, is the collection of communication channels. Assume that only links are apt to fail. Also assume independence of link failures.

To find our performance or reliability measure for any packet-switched network subject to random failure, our algorithm implies performing the following steps:

- 1- Using a routing algorithm the link flows  $\lambda_i$  are found by applying Eq. 3.8 for the no failure and for each of the fail states. We will consider the single failure states, i.e., one link failing at a time, and the two failure states, i.e., two links failing at a time.
- 2- The link flows  $\lambda_i$  are used to find the network average delay for each of the states: no failure state, single failure states, two failure states, using Eq. 3.18.
- 3- The network average delay over all states will be found.



Also, the network average delay can be considered as a random variable which varies depending on which state the network is in (Li et al.,1984). We can study the distribution of network average delay, for a network subject to random failure.

To find the link flows for each of the states, only the  $m$  most probable states will be considered.

The network average for every state is enumerated using Kleinrock's delay formula. i.e., Eq. 3.18 is evaluated.

With respect to the delay evaluation for every failure state, this evaluation will be greatly affected by the routing and the rerouting policy. We assume that the traffic otherwise lost in the failed link will be rerouted instead.

When link availabilities are high, there is no need to use the Li and Silvester's algorithm for partial state enumeration which generates the  $m$  most probable states, and orders them according to their probability of occurrence. Since the no-failure and single failure states are the most probable ones and cover most of the state space as will be shown in chapter five. The above method works well when the network links have high availability, and the network itself is small. But if the link availability decreases, or if the network is large, more than one failure may occur at the same time. To increase the accuracy of our work we will consider the

states where one, or at most two links will fail at a time, and apply it to networks of various sizes.

## 4.2 Routing

In order to explain our model further, some definitions concerned with routing must be made.

The real function of the network layer is routing packets from the source machine to the destination machine. The routing algorithm is that part of the network layer software responsible for deciding which output line an incoming packet should be transmitted on (Tanenbaum,1996).

In packet-switched networks, if the network uses datagrams, this decision is made each time a new data packet arrives. In virtual-circuit connections, routing decisions are made only when a new virtual circuit is being set up. After that, data packets just follow the previously established route, i.e. a route remains in force for an entire user session.

Routing algorithms can be grouped into two major classes: nonadaptive and adaptive. Nonadaptive algorithms do not base their routing decisions on measurements or estimates of the current traffic and topology, instead, the choice of the route to use to get from  $i$  to  $j$  is computed in advance, off-line, and downloaded to the IMPs when the network is booted (Static routing).

Adaptive algorithms, attempt to change their routing decisions to reflect changes in topology and the current traffic. Adaptive algorithms can be further subdivided into centralized routing, where information collected from the entire subnet is used to make routing decisions, (global algorithms). Another type of routing is where the local algorithms run separately on each IMP and only use information available there, such as queue lengths, (isolated algorithms). The third class of algorithms uses a mixture of global and local information. They are called distributed algorithms.

#### **4.2.1 Shortest Path Routing**

The idea of this technique is to build a graph of the subnet, with each node of the graph representing an IMP, or switching center and each link of the graph representing a communication line. To choose a route between a given pair of IMPs, the algorithm just finds the shortest path between them (Tanenbaum,1996).

The shortest path problem is to find a minimum length. This length or weight of a path could represent cost, delay, distance, bandwidth, average traffic, mean queue, etc...

As another example, if  $p_{ij}$  is the probability that a given link  $(i, j)$  in a network is usable, and each link is usable independently of all other links, then finding the shortest path between  $i$  and  $j$  with link

distances  $-\ln p_{ij}$  or,  $-\log p_{ij}$  is equivalent to finding the most reliable path from  $i$  to  $j$  (Bertsekas et al.,1987), (Cravis,1981).

The graph is then labeled with this weighting function. The algorithm used would then compute the “shortest” path according to any one of the above criteria.

#### 4.2.2 Dijkstra’s Algorithm

Several algorithms for computing the shortest path between two nodes of a graph  $G(N,L)$  are known. Dijkstra’s algorithm is adapted to centralized computation, and is probably one of the most efficient algorithms, and is still used today in the ARPANET.

Given a distance matrix  $S$  for a graph  $G(N, L)$ , the problem is that of finding the shortest path from a vertex, or node  $n_s$ , to another vertex, or node  $n_t$  (Dijkstra,1959).,  $n_s$  means the source node, and  $n_t$  means the destination node.

The set of vertices  $N$  is divided into two subsets  $T$  and  $W$ . The subset  $T$  is a vector containing vertices that have permanent labels associated with them. A permanent label of a vertex is the minimum distance of the vertex from  $n_s$ . This distance is stored in the vector  $PL$ . Thus associated with  $T$  is a distance vector  $PL$ . Therefore, Dijkstra’s algorithm is considered to be a labeling algorithm.

The set  $W$  is simply  $(N-T)$ . The vector  $P$  contains the vertex adjacent to  $T$  along the minimum path. The set  $TL$  defines the temporary labels associated with vertices in  $W$ . The distance matrix  $S$  is defined to consist of nonnegative entries,  $l_{ij}$ , as discussed in chapter five.

The steps involved in Dijkstra's algorithm are given below:

**Step 1:** Initially  $T = \{n_s\}$ ,  $PL = \{0\}$ , and  $P = \{0\}$ .

A temporary label of  $\infty$  (infinity) is associated with all vertices in  $W$ .

**Step 2:** Using the distance matrix ( $S$ ), the vertices in  $W$  that are adjacent to any vertex in  $T$  are determined. Each of these vertices is assigned a temporary label equal to the distance of that vertex from  $n_s$ . The distance of vertex  $n_i$  in  $W$  adjacent to  $n_j$  in  $T$  is its new temporary label given by:

Temporary label of  $n_i = \min_j (\text{permanent label of } n_j + l_{ji})$ .

**Step 3:** The smallest temporary label is made permanent. The corresponding vertex  $n_i$  is transferred from  $W$  to  $T$ . The vertex  $n_j$  is included in  $P$  from  $T$  that is adjacent to it.

All temporary labels are reset to  $\infty$  (infinity).

Steps 2 and 3 are repeated until  $n_t$  is included in T or there are no vertices in W that are adjacent to those in T.

**Step 4:** if  $n_t$  is included in T Step 5 is performed, otherwise the desired path does not exist. Stop.

**Step 5:** The permanent label of  $n_t$  gives the distance of the shortest path from  $n_s$  to  $n_t$ . To find the shortest path we start from  $n_t$  in T, check the corresponding vertex  $n_m$  in P. Starting from  $n_m$  in T the preceding vertex  $n_n$  in P is checked.

This process is repeated until  $n_s$  is checked. The checked sequence is the path from  $n_s$  to  $n_t$ .

### 4.2.3 Multipath Routing

In the previous shortest path routing algorithm, it was assumed that there is a single “best” or “shortest” path between any pair of nodes and that all traffic between them should use it. In many networks, there are several paths between pairs of nodes that are almost equally good. Better performance can frequently be obtained by splitting the traffic over several paths, to reduce the load on each of the communication lines (Tanenbaum,1996). The technique of using multiple routes between a

single pair of nodes is called multipath routing or sometimes bifurcated routing.

Multipath routing is applicable both to datagram networks and virtual circuit networks. For datagram networks, when a packet arrives at an IMP for forwarding, a choice is made among the various alternatives for that packet, independent of the choices made for other packets to that same destination in the past.

For virtual circuit networks, whenever a virtual circuit is set up, a route is chosen, but different virtual circuits are routed independently.

In multipath routing, each IMP maintains a table with one row for each possible destination IMP. A row gives the best, second best, third best, etc... outgoing line for that destination.

An advantage of multipath routing is the possibility of sending different classes of traffic over different paths. For example, a connection between a terminal and a remote computer that consists of short packets that must be delivered quickly could be routed via terrestrial lines, whereas a long file transfer requiring high bandwidth could go via a satellite link. This method ensures that the file transfer gets the high bandwidth needed, and it also prevents the short terminal packets from being delayed behind a queue of long file transfer packets.

Multipath routing could also be used to improve reliability. For example, if the routing tables contain  $n$  disjoint routes between each pair of

IMPs, then the network can withstand the loss of  $n-1$  lines without being disconnected.

Therefore, a variant of the shortest path problem is that of finding the  $k$  shortest paths between source and destination (Dreyfus,1969), (Yen,1971), which has been treated extensively in the literature.

One simple way to make sure that all the alternative routes are disjoint, is first to compute the shortest path between source and destination. Then to remove from the graph all the nodes and links used on the shortest path and to compute the shortest path through the new graph. This algorithm ensures that IMP or line failures on the first path will not also bring down the second one. By removing the second path from the graph as well, a third path can be computed that is completely independent of the first two (Tanenbaum,1996).

#### **4.2.4 Centralized Routing**

All routing algorithms require information about the network topology and traffic to make decisions (Tanenbaum,1996).

In real networks IMPs and lines go down and come back up again. Also the traffic varies wildly throughout the day. Therefore, some techniques for building the routing tables is needed.

When centralized routing is used, routing tables are built in a central location called RCC (Routing Control Center). Periodically, each IMP



sends status information to the RCC (i.e., a list of its neighbours that are up, current queue lengths, amount of traffic processed per line since the last report, etc.). The RCC collects all this information, and then, based upon its global knowledge of the entire network, computes the optimal routes from every IMP to every other IMP. From this information it can build new routing tables and distribute them to all the IMPs.

Centralized routing may seem attractive: since the RCC has complete information, it can make perfect decisions. Another advantage is that it relieves the IMPs of the burden of the routing computation.

However there are serious and fatal drawbacks of centralized routing: One of them is that if the network is to adapt to changing traffic, the routing calculation will have to be performed fairly often for a very large network, which will take many seconds.

Another serious problem is the vulnerability of the RCC. If it goes down or becomes isolated by line failures, the whole network will be in trouble. If a second machine is used as a backup, a large computer would be wasted. Also a method must be used to ensure which machine runs the network, primary or secondary.

When distributing the routing tables, the IMPs that are close to the RCC will get their new tables first and will switch to the new routes before the distant IMPs have received their tables. Inconsistencies may arise, and

packets delayed, including the packets containing the new routing tables for the distant IMPs.

If alternate routes are not computed by the RCC, the loss of even a single line or IMP may cut off some IMPs from the RCC.

Due to the flow of routing information, heavy traffic will be concentrated on the lines leading into the RCC. Also the vulnerability of those lines is increased.

#### **4.2.5 Distributed Routing**

This class of routing was originally used in the ARPANET. In this algorithm each IMP periodically exchanges explicit routing information with each of its neighbors. Typically, each IMP maintains a routing table indexed by, and containing one entry for, each other IMP in the network. This entry contains two parts: the preferred outgoing line to use for that destination, and some estimate of the time or distance to that destination (Tanenbaum, 1996). The metric used might be number of hops, estimated time delay in milliseconds, estimated total number of packets queued along the path, excess bandwidth etc...

The IMP is assumed to know the "distance" to each of its neighbors. If the metric is hops, the distance is just one hop. The newer link state algorithms can implement up to eight metrics in their route selection process, and only broadcast that part of their routing tables which has

changed and only when required (Fujitsu Ltd.,1994). If the metric is queue length, the IMP simply examines each queue. If the metric is delay, the IMP can measure it directly with special “echo” packets that the receiver just time stamps and sends back as fast as it can.

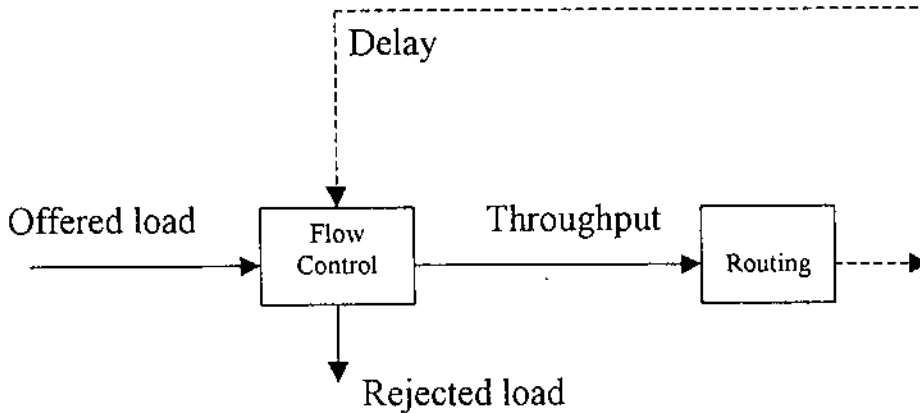
#### **4.2.6 Main Issues in Routing**

The two main functions performed by a routing algorithm are the selection of routes for various source destination pairs and the delivery of messages to their correct destination once the routes are selected (Bertsekas et al.,1987).

The second function is accomplished through a variety of protocols and data structures (known as routing tables). The first function which is the selection of routes affects the network performance.

The two main performance measures that are substantially affected by the routing algorithm are: throughput (quantity of service), and average packet delay (quality of service). Routing interacts with flow control in determining these performance measures. Flow control, relates to the point-to-point traffic between a given sender and a given receiver (Tanenbaum,1996). Its job is to make sure that a fast sender cannot continually transmit data faster than the receiver can absorb it.

The interaction of routing and flow control can be modeled by means of a feedback mechanism as shown in Fig. 4.1. As good routing keeps delay low, flow control allows more traffic into the network.



**Fig. 4.1 Interaction of routing and flow control(Bertsekas et al.,1987)**

When the traffic load offered by the external sites (hosts) to the network is relatively low, it will be fully accepted into the network, i.e.,

$$\text{Throughput} = \text{Offered load}$$

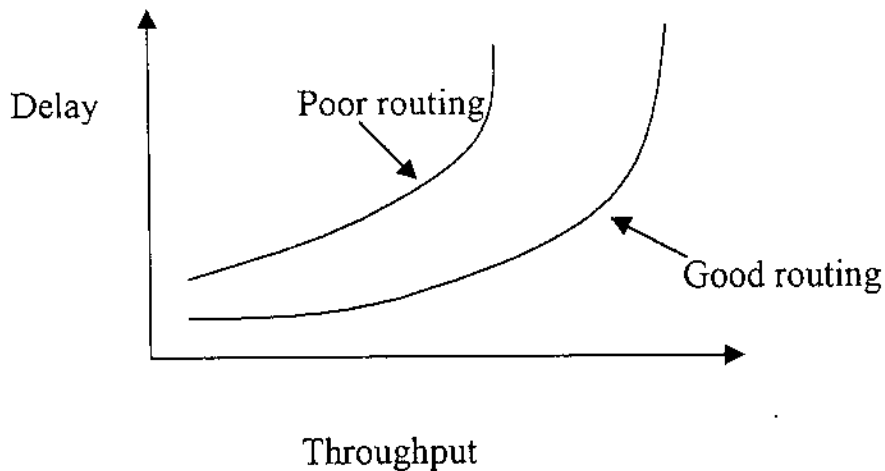
When the offered load is excessive, a portion will be rejected by the flow control algorithm and

$$\text{Throughput} = \text{Offered load} - \text{Rejected load}$$

The traffic accepted into the network will experience an average delay per packet that will depend on the routes chosen by the routing algorithm. However, throughput will also be greatly affected (indirectly) by the routing algorithm because typical flow control schemes operate on the

basis of striking a balance between throughput and delay, i.e, they start rejecting offered load when delay starts getting excessive.

Therefore, as the routing algorithm is more successful in keeping delay low, the flow control algorithm allows more traffic into the network. While the precise balance between delay and throughput will be determined by flow control, the effect of good routing under high offered load conditions is to realize a more favorable delay-throughput curve along which flow control operates. An example is shown in Fig. 4.2 (Bertsekas et al.,1987).



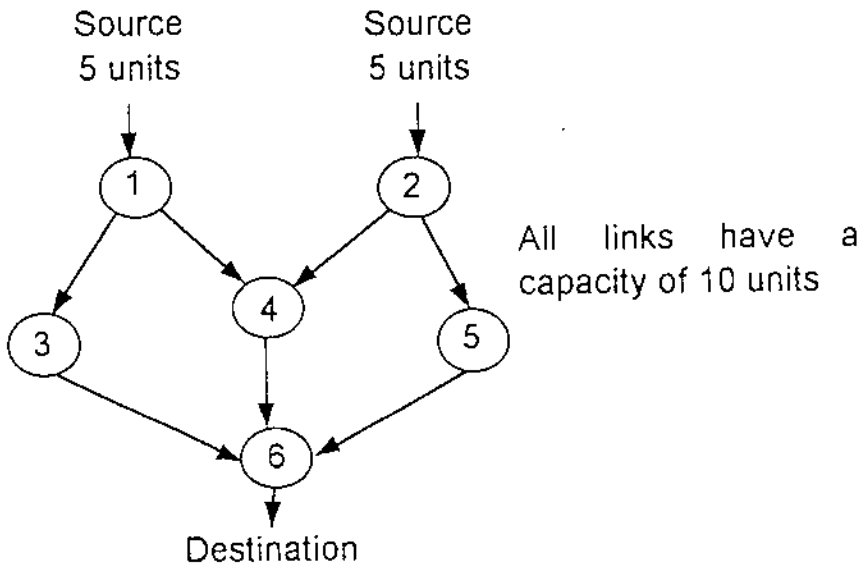
**Fig. 4.2 Delay-throughput operating curves for good and bad routing (Bertsekas et al.,1987)**

The following examples illustrate the above discussion:

In the network of Fig. 4.3 all links have capacity of 10 units. There is a single destination (node 6) and two sources (nodes 1 and 2). The offered load from each of nodes 1 and 2 to node 6 is 5 units. Here, the offered load

is light and can be easily accommodated with small delay by routing along the left-most and right-most paths,  $1 \rightarrow 3 \rightarrow 6$  and  $2 \rightarrow 5 \rightarrow 6$ , respectively.

If instead, however, the routes  $1 \rightarrow 4 \rightarrow 6$  and  $2 \rightarrow 4 \rightarrow 6$  are used, the flow on link (4,6) will equal capacity, resulting in very large delays.



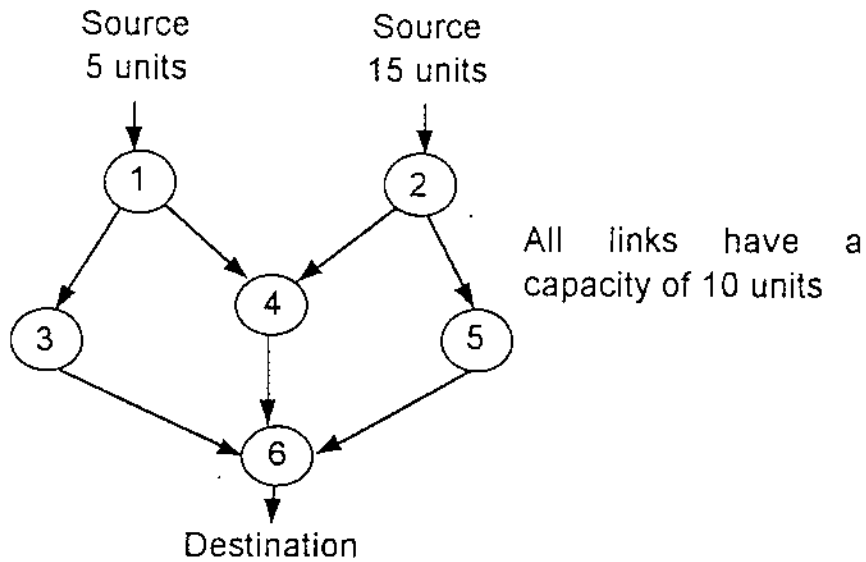
**Fig. 4.3 Routing example**

If all traffic is routed through the middle link (4, 6), congestion occurs. If instead paths  $(1 \rightarrow 3 \rightarrow 6)$  and  $(2 \rightarrow 5 \rightarrow 6)$  are used, the average delay is small.

For the same network, suppose that the offered loads at nodes 1 and 2 are 5 and 15 units, respectively as shown in Fig. 4.4. If routing from node 2 to the destination is done along a single path, then at least 5 units of offered load will have to be rejected since all path capacities equal 10. Thus, the total throughput can be no more than 15 units. However if the

traffic originating at node 2 is evenly split between the two paths  $2 \rightarrow 4 \rightarrow 6$  and  $2 \rightarrow 5 \rightarrow 6$ , while the traffic originating at node 1 is routed along  $1 \rightarrow 3 \rightarrow 6$ . Then, the traffic arrival rate on each link will not exceed 75% of capacity, the delay per packet will be reasonably small, and (given a good flow control scheme) no portion of the offered load will be rejected.

It is seen that when the offered loads at nodes 1 and 2 are both large, the maximum total throughput that this network can accommodate is between 10 and 30 units, depending on the routing scheme. This also shows that to achieve high throughput, the traffic of some source-destination pairs may have to be divided among more than one route.



**Fig. 4.4 Routing example**

The input traffic can be accommodated with multiple path routing, but at least 5 units of traffic must be rejected if a single-path routing is used.

The above discussion is shown in Kleinrocks formula Eq. 3.18, where we must have  $\mu C > \lambda$ , in any practical system. Otherwise, the system is unstable in the sense that the average delay, and hence the average time in the system, increases beyond all bounds.

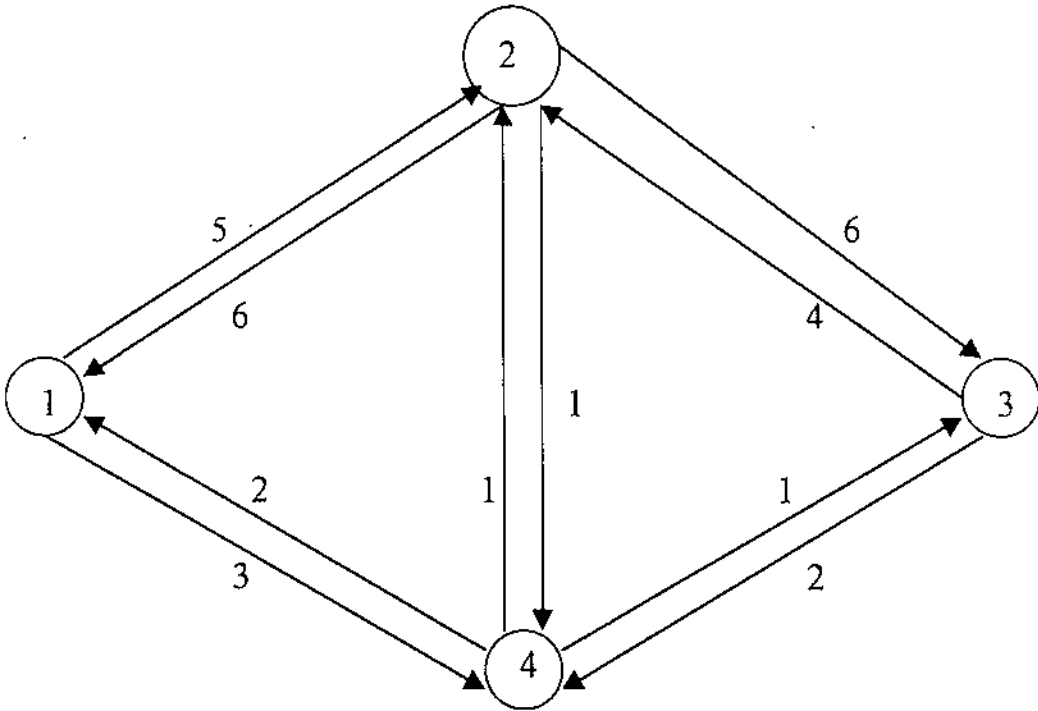
#### 4.2.7 Routing in Practice

As an example of routing, we will consider the ARPANET. The routing algorithm used is known as the shortest path first algorithm. There are a number of such algorithms but the one used is the ISO Dijkstra's algorithm.

The routing algorithm is a distributed, adaptive algorithm. In this algorithm, each node maintains a database describing the delay on each network line. A shortest-path computation is run in each node which explicitly computes the minimum-delay paths (based on the delay entries in the database) from that node to all other nodes in the network. The average delay on each network line is measured periodically by the nodes attached to the lines. These measured delays are broadcast to all network nodes, so that all nodes use the same database for performing their shortest-path computations (McQuillan,1995).

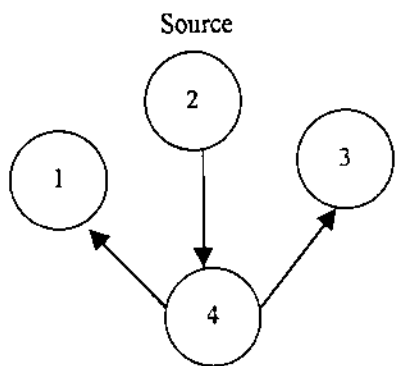


The routing tables of an ARPANET node contain only the first outgoing link of the shortest path from the node to each destination, as shown in Fig. 4.5. When the node receives a new packet, it checks the packet's destination, it consults its routing table, and it places the packet in the corresponding outgoing link queue.

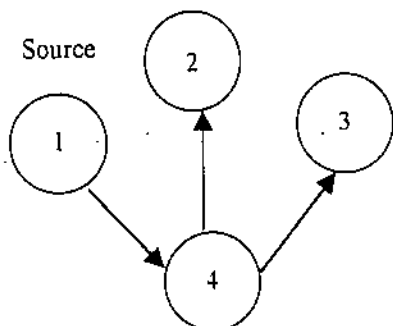


The numbers next to the links are link lengths (or weights).

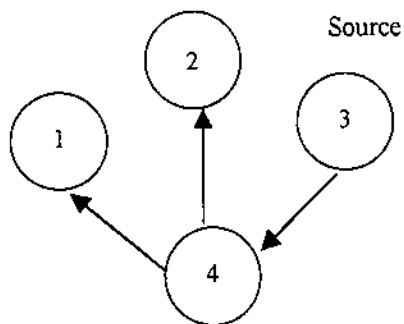
**Fig. 4.5a ARPANET nodes.**



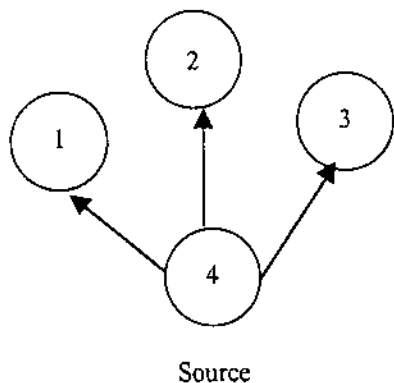
Destination	Out link
1	(2,4)
3	(2,4)
4	(2,4)



Destination	Out link
2	(1,4)
3	(1,4)
4	(1,4)



Destination	Out link
1	(3,4)
2	(3,4)
4	(3,4)



Destination	Out link
1	(4,1)
2	(4,2)
3	(4,3)

**Fig. 4.5b** Routing tables maintained by the nodes in the ARPANET algorithm. Each node calculates a shortest path from itself to each destination, and enters the first link on that path in the routing table.

The algorithm used in the ARPANET is considered of the link state type. Where the state of the link, such as bandwidth or measured delay, is taken into account when making routing decisions (Tanenbaum,1996), (Fujitsu Ltd.,1994).

The five points which make up link state routing are:

- 1- Each Switching Center, IMP, or Router must discover its neighbors and learn their network addresses. The neighbors may be other routers or local area networks, each of which is represented as a node.

The discovery is done by sending a special HELLO packet on each point-to-point line. The router on the other end is expected to send a reply telling who it is.

- 2- Each IMP must measure the delay or cost to each of its neighbors. This is done by sending a special ECHO packet, which the other end is required to send back immediately. The time required to send the packet and receive a reply is called the round-trip time. The time used is the round-trip time divided by two. Sometimes an average of several tests is used.

- 3- Each IMP must construct a packet telling all it has just learned. This is called a link state packet. This packet contains the identity of the sender, followed by a sequence number and age, and a list of neighbors. For each neighbor, the delay to that neighbor is given.

- 4- The IMP must send this packet to all other neighbors. The technique used for distributing packets is flooding. This is done by giving each packet a sequence number that is incremented for each new packet sent. The IMPs keep track of all the (source router, sequence no.) pairs they receive. When a new link state packet comes in, it is checked against the list of packets already received. If it is new, it is forwarded on all lines except the one it arrived on. If it is a duplicate, it is discarded. If a packet with a sequence number lower than the highest one seen so far arrives, it is rejected as being obsolete.
- 5- Finally, the shortest path is computed to every other neighbor. When an IMP has accumulated a full set of link state packets, the entire subnet graph can be constructed because every link is represented. As described, Dijkstra's algorithm can now be run locally to construct the shortest path to all possible destinations.

### **4.3 Our Reliability Algorithm**

Now we are in a position to state the steps involved in our reliability algorithm intended to calculate the reliability measure of interest defined as the network average delay based on Eq. 3.18. This can be applied to any network subject to random failure. Before stating the main steps to be

undertaken, a routing algorithm is needed, and it involves the following features:

#### 4.3.1 New Routing Algorithm

Our routing algorithm is a distributed algorithm in the sense that all available information on the whole network is used at each node in order to make routing decisions.

Assuming only links are apt to fail, each state will represent a different network to which the routing problem is applied. Associated with this approach is the assumption that the traffic in the lost link will be rerouted in the new network. To clarify this, assume that the original network of  $N$  nodes and  $L$  links is represented by the graph  $G(N, L)$ . If for example, one link of this network fails, it is removed from the graph and we get a new network with  $N$  nodes and  $L'=L-1$  links, represented by the graph  $G(N, L')$  to which the routing problem is again applied.

The routing algorithm is based on shortest path calculations only between nodes which are exchanging traffic, and not between each pair of nodes, and not between a central node and every other node in the network as in classical approaches (Cravis,1981), (Schwartz et al,1980).

The shortest path calculations are based on Dijkstra's algorithm. The metric or weighting function used in our calculation is availability associated with each link. Assuming that the probability of a route is the

product of link probabilities (Cravis,1981), then the weight of a link between nodes  $i$  and  $j$ ,  $w_{ij} = -\log p_{ij}$ , where  $p_{ij}$  is the availability of link  $ij$ , or the probability of it being operational. Thus, in our shortest path calculations, we calculate the most reliable path for the traffic between a pair of nodes.

As discussed in section 4.2.3, sometimes it is necessary to use multipath routing. In our algorithm this is achieved by first calculating the first shortest, or most reliable path, then we remove the links associated with the first shortest path to get a new graph and to calculate the second shortest path. In this way we ensure that the first and second shortest path are link disjoint. This is done in an attempt to increase the reliability of the network.

In section 4.2.6 we considered the relation between throughput and network average delay. In chapter five we applied several different loads to a test network and studied the effect on the average network delay. From those conducted tests, it was seen that when the load increased, and single path routing was used, the load in some of the links exceeded the available capacity, which caused infinite delay conditions on those links, and infinite average network delay when applying Eq. 3.18. To overcome this problem, whenever the load of a link exceeds its capacity our algorithm uses the second shortest path to route traffic.

If the overload condition prevails even after using secondary paths, the excess traffic is rejected and the load of the link is made equal to its capacity, resulting in an infinite delay condition. Which in reality would be a bottleneck associated with this link.

The routing algorithm described above was used in association with the reliability algorithm.

#### 4.3.2 The Reliability algorithm

The algorithm can be described in the following steps:

**Step 1:** The network states are generated in their order of occurrence, that is, first of all the no failure state is considered, where the original network with all of its links connected is used to perform our calculations.

... Then the single failure states are generated. Single failure states are states where exactly one link has failed at a time. For a network of  $L$  links, the single failure states to be considered are  $L$  (the number of links).

The last state to be considered are the two failure states, that is: the states where exactly two links at a time have failed. For the same  $L$  link network, the two failure states count up to  $\frac{L(L-1)}{2}$

(Sutari,1996). Therefore, the total number of states to be

generated is  $1+L+\frac{L(L-1)}{2}$ . These are the most probable states to

occur in a high availability network as will be shown in chapter 5.

**Step2:** For each of the generated states our algorithm is applied, between any two nodes exchanging traffic in order to calculate the shortest path, and the second shortest path between them.

The weight or length measure used to perform our shortest path calculations is availability (usability), or probability of links remaining operable. So our shortest path is the most reliable path, or the path with the highest probability of being operable.

Sometimes as in our case, we have to consider the second shortest path to be used in order to increase reliability. This second shortest path which is link disjoint from the first, serves as a backup route, in an attempt to minimize the average network delay, which may be experienced by messages in a network especially when failure occurs and overload conditions happen (Gavish et al.,1992).

**Step 3:** Applying routing, using the shortest path, and second shortest path calculations described above, the link flows  $\lambda_i$  are calculated,  $i=1, 2, \dots, L$ .



When a link experiences an overload condition, i.e.,  $\lambda_i > \mu C_i$ , the second shortest path is used instead. If the overload condition persists the excess traffic is rejected, and maximum available capacity is used, i.e.,  $\lambda_i = \mu C_i$ , resulting in an infinite delay condition over this line.

**Step 4:** For each of the above generated states  $S_k$ , and calculated link flows  $\lambda_i$ , the network average delay for each link  $T_i$ ,  $i=1, 2, \dots, L$ , is calculated using Eq. 3.13. Using Eq. 3.12,

$$T(S_k) = \sum_{i=1}^L \frac{\lambda_i}{\gamma} T_i$$

gives the network average delay for the particular state  $S_k$  considered.

**Step 5:** Bounds are calculated. That is an upper and lower bound on the expected value of the average delay over all states is calculated using Eq. 3.20

$$T_L(m) = \sum_{k=1}^m P(S_k) T(S_k) + \left(1 - \sum_{k=1}^m P(S_k)\right) T(S_0)$$

$$T_U(m) = \sum_{k=1}^m P(S_k) T(S_k) + \left(1 - \sum_{k=1}^m P(S_k)\right) T(S_{2^n})$$

Those bounds are calculated by using the delays of each state generated in step 4  $T(S_k)$ , and by calculating the probability of occurrence of each state  $P(S_k)$ , as described by Eq. 3.1.

**Step 6:** We can also calculate an estimator for the network average delay  $\tilde{T}_m$ , using Eq. 3.21,

$$\tilde{T}_m = \frac{\sum_{k=1}^m P(S_k)T(S_k)}{\sum_{k=1}^m P(S_k)}$$

**Step 7:** Also the distribution of network average delay can be calculated using Eq. 3.22,

$$\tilde{F}_m(t) = \sum_{\substack{S_k : T(S_k) \leq t \\ k \leq m}} P(S_k)$$

The results obtained using steps 6 and 7 are plotted as will be shown in chapter 5.

# Chapter Five

## Results and Conclusions

### 5.1 Introduction

In this chapter our results will be presented, showing the proposed reliability measure as the average network delay for a network subject to random failure by taking into account the most probable states of the network.

The routing technique applied is dependent on shortest path routing, where shortest refers to the most reliable path, under normal loading conditions, and the second most reliable path, which serves as a backup path in the case of an overload condition of a link.

Several networks will be tested, different loading conditions will be considered and the availability of links changed. But first of all our approach will be tested to see if the number of states considered is enough to get an accurate performance measure.

Matlab was used for the Reliability algorithm, and for plotting our results. The machine used to conduct our tests was a 486 DX personal computer.

### 5.1.1 Generating the states

As stated earlier, our network could be in any of a number of states. Only the most probable states to occur will be considered. That is, the no failure state, i.e., the original network with all links operating, the single failure states, i.e., states where one link fails at a time, where those states add up to  $L$ , the number of links, and finally the two failure states, i.e., states where any two links fail together. The two failure states add up to  $\frac{L(L-1)}{2}$ .

In generating the above states, we have used the connection matrix of the network. An example of a connection matrix is given in Appendix A4. For a graph  $G(N, L)$ , the connection matrix ( $A$ ) is a  $n$ -by- $n$  symmetric matrix where  $A=[a_{ij}]$

$$a_{ij} = \begin{cases} 1 & \text{if a link exists between node } i \text{ and } j \\ 0 & \text{otherwise} \end{cases}$$

The reliability algorithm tests the connection matrix, and checks its entries, if an entry  $a_{ij}=1$ , this means a link exists between node  $i$  and  $j$ . In order to generate a failure state, the corresponding entry in the distance matrix  $S=[l_{ij}]$  is set to  $\infty$  (infinity). An example of a distance matrix is given in Appendix A4. The distance matrix is a  $n$ -by- $n$  matrix where  $S=[l_{ij}]$

$$l_{ij} = \begin{cases} \text{the weight of link } ij \text{ if a link exists between } i \text{ and } j \\ \infty & \text{otherwise} \end{cases}$$

The single failure, and two failure states are generated in the manner described above, each time a state is generated a different distance matrix is obtained on which the shortest path calculations can be made, link flows computed, and average network delay obtained.

### 5.1.2 Shortest path calculations

In performing our shortest path, or most reliable path calculations, the input file of Appendix A4 is used.

This file contains the network data necessary to perform our calculations, like  $n$ : No of nodes,  $S$ : The distance matrix,  $g_a$ : The traffic matrix, where  $g_a = [g_{a_{ij}}]$

$$g_{a_{ij}} = \text{amount of traffic between node } i \text{ and } j (\gamma_{ij})$$

The capacity of the links  $C$ , and the packet length  $1/\mu$ .

The shortest path algorithm examines the traffic matrix to see which two nodes exchange packets in order to calculate the most reliable and second most reliable path between them to use it to send the packets across. The algorithm used for our shortest path calculations is similar to Dijkstra's algorithm. The difference between our algorithm and Dijkstra's is that in Dijkstra's algorithm, the shortest path from a root node to all other nodes is computed. In contrast, our algorithm reduces calculation time by performing shortest path calculations only on those nodes which exchange traffic.

### 5.1.3 Link flows

After the most reliable paths for the packets have been computed, the link flows  $\lambda_i$  for  $i=1, \dots, L$  are obtained by applying Eq. 3.8 on each of the links:

$$\lambda_i = \sum_j \sum_{i,j: C_i \in \pi_{ij}} \gamma_{ij}$$

### 5.1.4 Average delay

After calculating the link flows for a particular state, those flows are used to calculate the average time of the system for this state by applying Eq. 3.18

$$T(S_k) = \sum_{i=1}^L \frac{\lambda_i}{\gamma} \left[ \frac{1}{\mu C_i - \lambda_i} \right] = \sum_{i=1}^L \frac{\lambda_i}{\gamma} T_i$$

where  $T_i = \frac{1}{\mu C_i - \lambda_i}$  is the delay on a particular link. This means that the average time of the system equals the sum of the delays on the links of that system.

### 5.1.5 Probability of States

Associated with each state is a probability of that state occurring, which depends on the availabilities of the links. The probability of occurrence of each of the states is calculated by applying Eq. 3.1.

$$P(S_k) = \prod_{i=1}^L p_i (q_i / p_i)^{T_i(S_k)}$$

The state probabilities are used to calculate the probability distribution function using Eq. 3.22. This is done by calculating the commulative probability  $F(S_k)$  of the average network delay for a particular state  $T(S_k)$ , and plotting the results. This is shown in Appendix A2. Examples of probability distribution functions are given in Section 5.2.

After calculating the probability of the states, A lower bound on the expected average delay over all states can be obtained using Eq. 3.20,

$$T_L(m) = \sum_{k=1}^m P(S_k)T(S_k) + \left(1 - \sum_{k=1}^m P(S_k)\right)T(S_0)$$

Finally, the average delay over all states is calculated by using Eq.3.21,

$$T_m = \frac{\sum_{k=1}^m P(S_k)T(S_k)}{\sum_{k=1}^m P(S_k)}$$

which is our performance measure of interest. The above calculations are given in Appendix A2. The results are given in section 5.3.

In section 5.4 the effect of link availability on the average network delay is studied. Section 5.5 shows the effect of the number of links. The calculations and plots are given in Appendix A3.

Appendix A4 shows a sample of an output file for Fig. 5.1 giving the shortest route weight matrix (S), the successor node matrix (R), the link flows  $\lambda_i$ , and network average delay  $T(S_k)$  for the no failure state.

## 5.2 The Most Probable States

If all states are to be considered we would take the no failure, single failure, two failures, three failures, ..., up to where all links of an L link network have failed.

To calculate the probability of a single state to occur we use Eq. 3.1

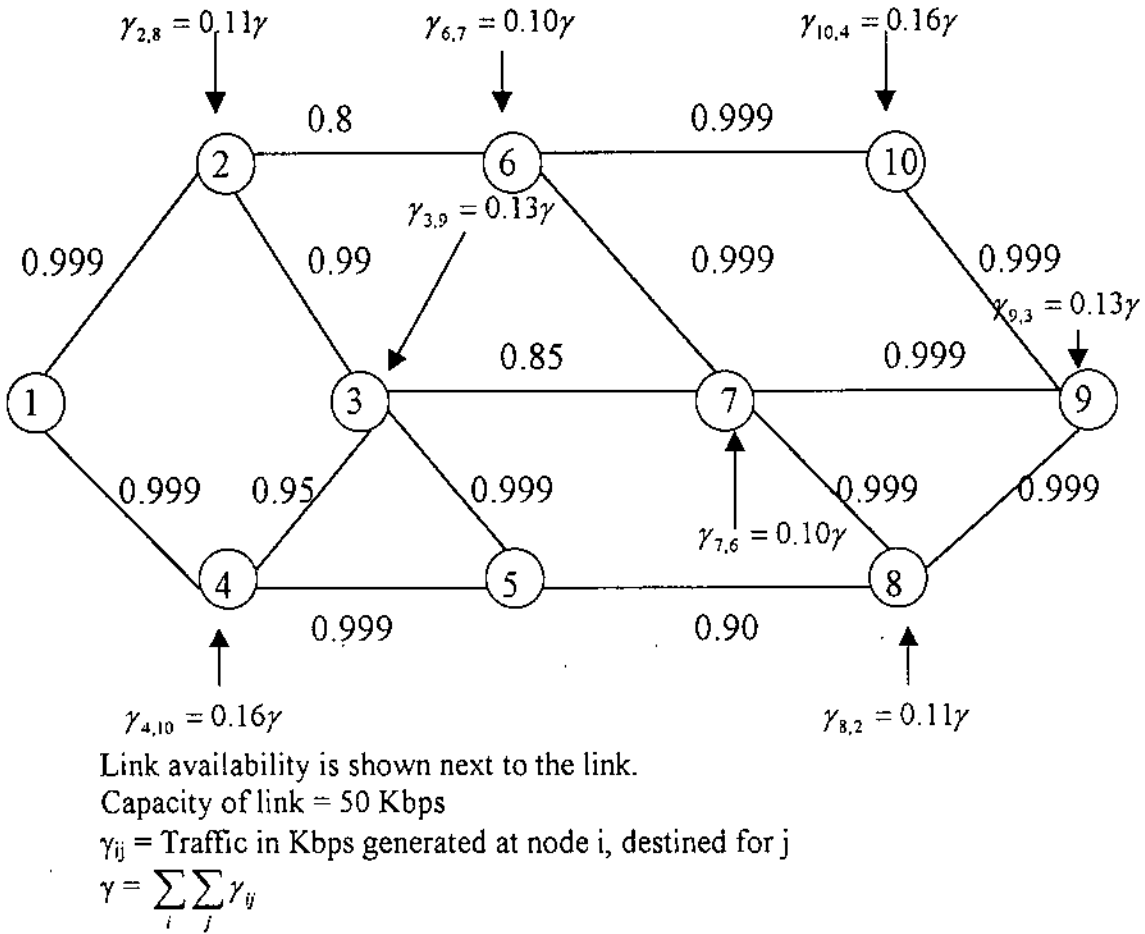
$$P(S_k) = \prod_{i=1}^n p_i (q_i / p_i)^{T_i(S_k)} = \left( \prod_{i=1}^n P_i \right) \left( \prod_{j=1}^n R_j^{T_j(S_k)} \right)$$

The sum of the probability of occurrences of all states adds up to one.

As shown in (Sanso et al.,1991) for small networks, the state with the highest occurrences probability is the no failure state, followed by the states where exactly one link fails. In (Sutari,1996) we saw, as the network increases in size, additional states have to be considered. For networks with high link availabilities, if we consider the no failure, single failure, and two failure states, the sum of the probability of occurrences of those states adds up to more than 0.95.

To show the validity of the above statements, the following network in Fig. 5.1 is considered, with 15 links and 10 nodes (Li et al.,1984).





**Fig. 5. 1 Communication network.**

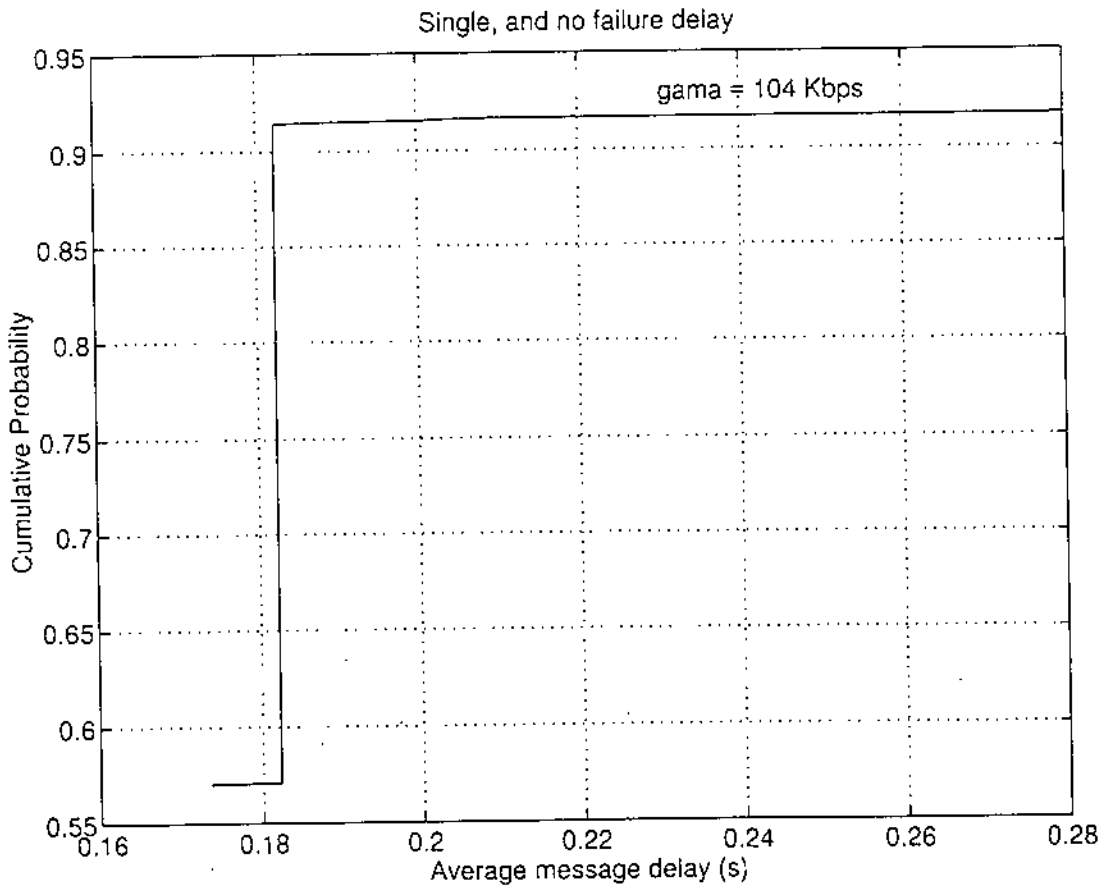
The network selected is a relatively reliable network, so that we will encounter fewer states where infinite delay occurs due to the network becoming disconnected.

First of all we applied a load of 104 Kbps to the network, and calculated the average delay using only the no failure and single failure states.

The probability distribution function for the network average delay is plotted in Fig. 5.2 using Eq. 3.22,

$$\tilde{F}_m(t) = \sum_{\substack{S_k : T(S_k) \leq t \\ k \leq m}} P(S_k)$$

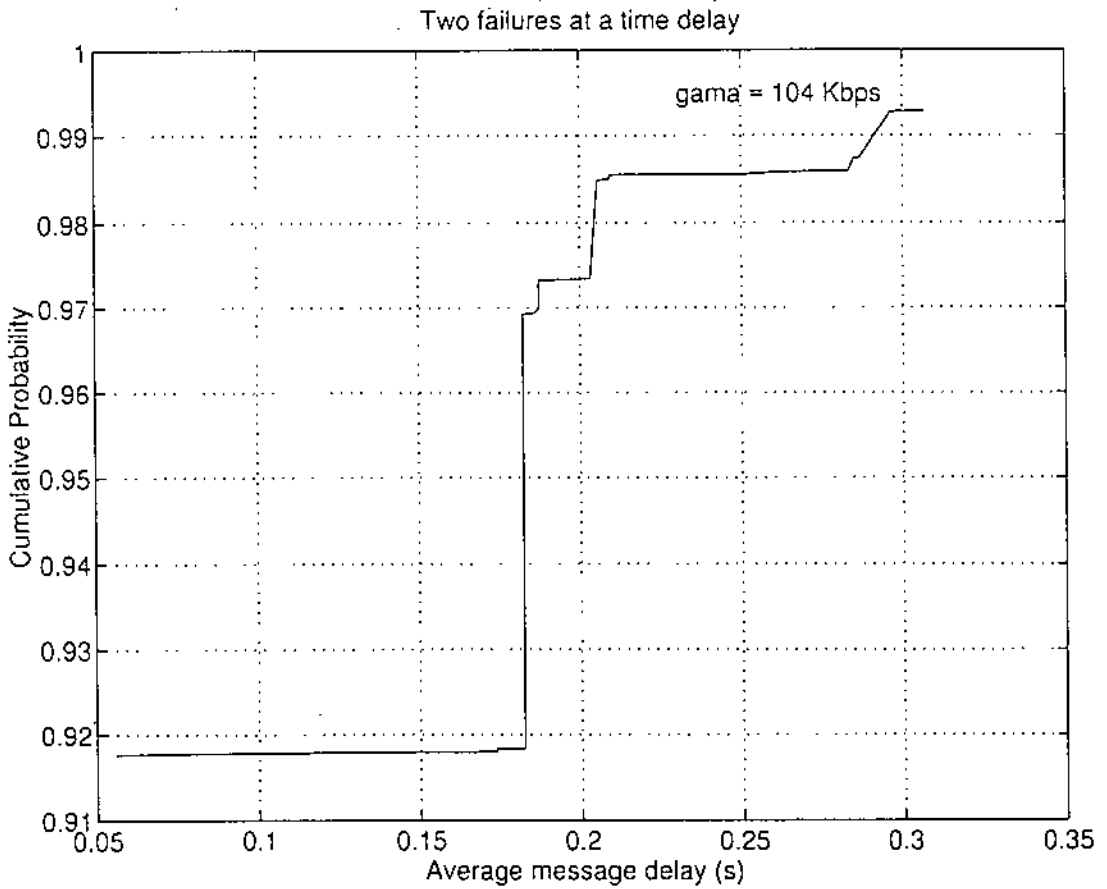
As stated earlier each point on the plotted curve represents the cumulative probability of the average delay of the particular state considered. If the whole state space in to be considered, the probability of the last state, i.e., the state with the highest delay should equal one.



**Fig. 5.2 Probability distribution function for the no failure and single failure states.**

We can see that the sum of probability of occurrences of the no and single failure states adds up to more than 0.9. This means in order to increase the accuracy of our work more states have to be considered.

Another possibility is only to consider the states where two failures at a time occur as shown in Fig. 5.3

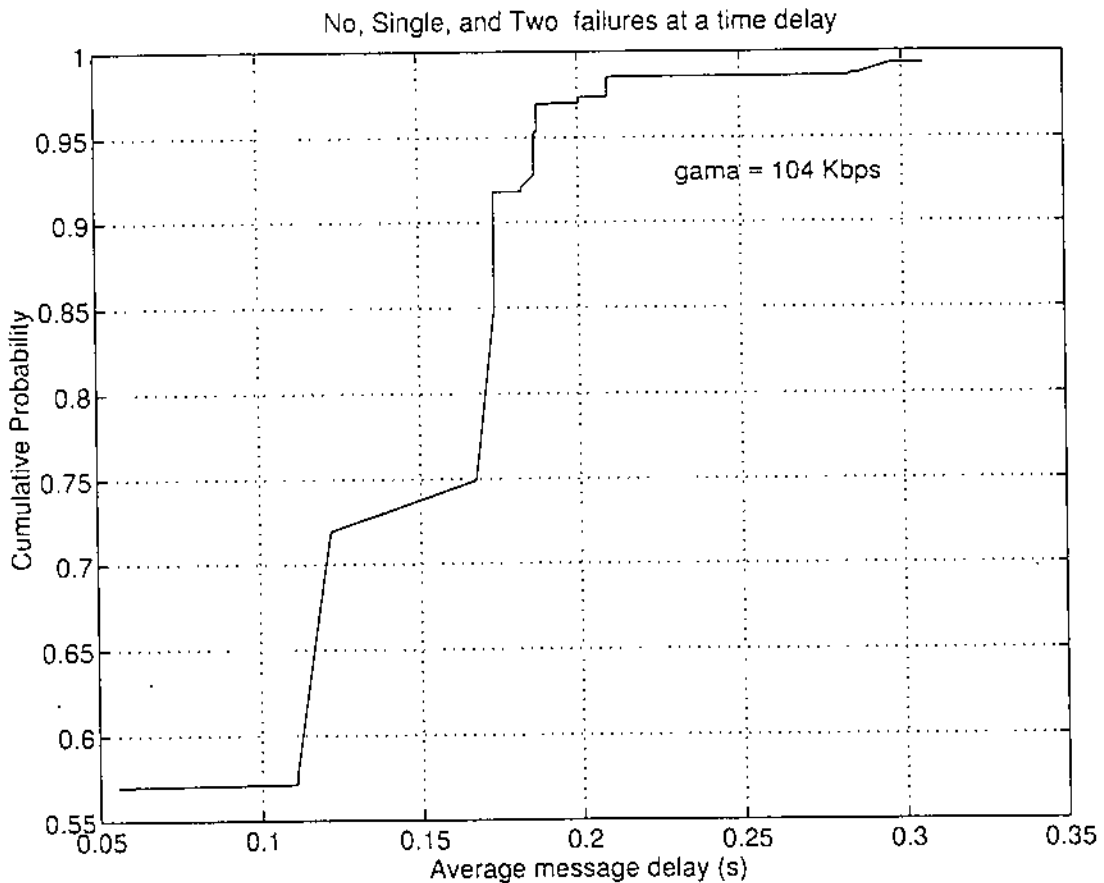


**Fig. 5.3 Probability distribution function for two failure states.**

As can be seen the average network delay is higher when two links at a time fail, also, the cumulative probability of the states is less than 0.95.

In Fig. 5.4 the no failure, single failure, and two failures at a time states have been considered.

This means in order to cover a high portion of the state space, i.e., more than 95%, the no, single, and two failure states are enough. Also, the number of states considered tend to give an accurate performance measure by using Eq. 3.21, when calculating the average of the delay over all states.



**Fig. 5.4 Probability distribution function for the no, single, and two failure states.**

To show the same results in table form, consider Table 5.1 showing the sum of the probabilities of no, single, and two link failures, of the network of Fig. 5.1.

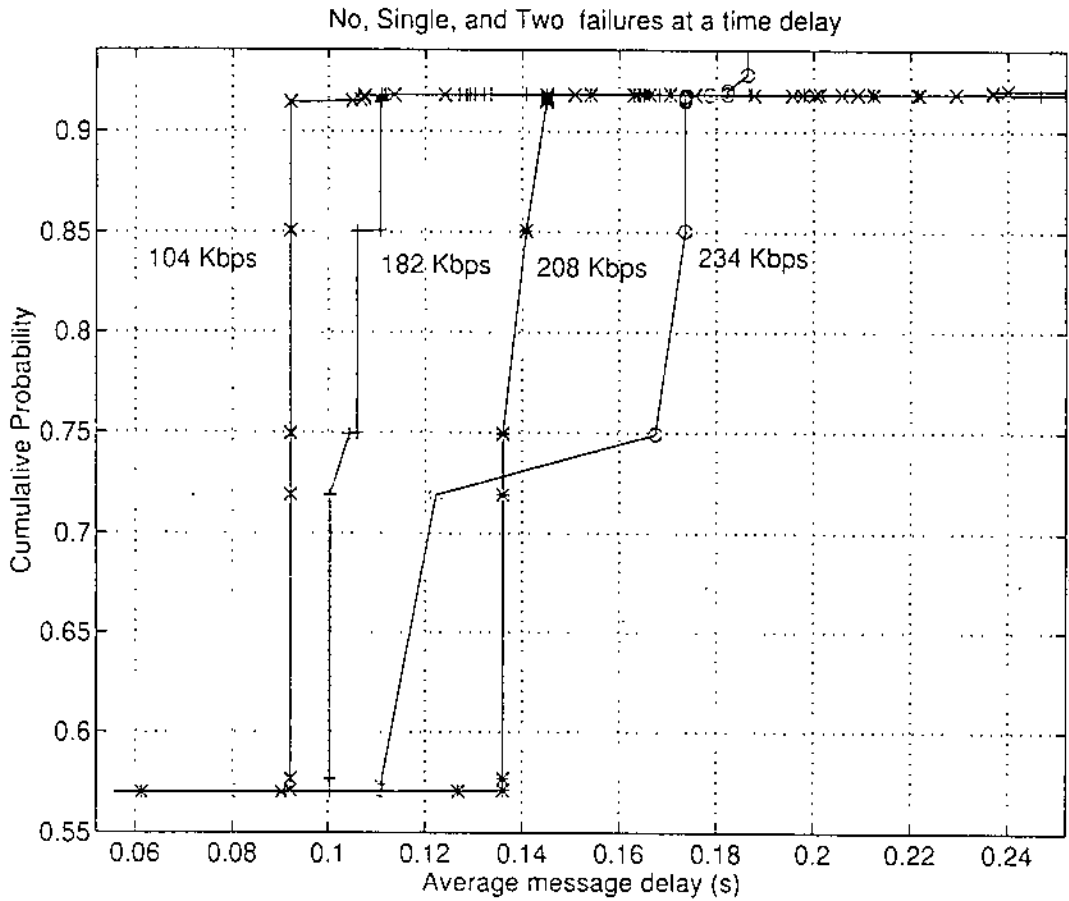
State	Probability
No failure ( $S_1$ )	0.5698
Single failures	0.3478
Two link failures	0.0751
No, Single, Two failures	0.9927

**Table 5.1:** Probability of states.

Thus 99% of all of the network states were considered in our approach.

As can be seen our approach covers a large fraction of the state space. The sum of the probability of occurrences of the states eventually adds up to nearly one.

Finally, the same approach is tested for different loading conditions, as shown in Fig. 5.5



**Fig. 5.5 Probability distribution functions using different loading conditions.**

We can see that in the event of heavy load like  $\gamma=234$  Kbps applied to the network, the delay increases when the applied load is increased, due to overload conditions and the use of secondary routes.

From the above conducted tests we can see that the no, single, and two failure states are enough to adequately describe the state space, and obtain accurate results for our performance measure.

### 5.3 Reliability Measure

As discussed previously our reliability or performance measure is network average delay as calculated using Kleinrock's delay formula 3.18 for a particular state

$$T = \frac{1}{\gamma} \sum_i^m \frac{\lambda_i}{\mu C_i - \lambda_i}$$

This performance measure is calculated for all the states considered.

The no failure, single failure, and two failures at a time states.

The reliability algorithm was applied to the network of Fig. 5.1 for a load of  $\gamma=104\text{Kbps}$ . Results are given in the Tables 5.2 and 5.3 below.

No. and Single Failure States	
Failed Link	Average delay in seconds
0	0.1823
1-2	0.1736
1-4	0.1736
2-3	0.1823
2-6	0.1823
3-4	0.1823
3-5	0.2798
3-7	0.1823
4-5	0.2782
5-8	0.2032
6-7	0.2004
6-10	0.1823
7-8	0.1823
7-9	0.1823
8-9	0.2093
9-10	0.1874

**Table 5.2** No and Single Failure Delay.

As can be seen from the results given above, the largest delay occurs when link 3-5 or link 4-5 fail. This is due to the fact that large amount of traffic used this link. By looking at the shortest route matrix given in Appendix A4 we see that the traffic between nodes 3 and 9 used link 3-5. Also the traffic between nodes 2 and 8, 10 and 4 used link 4-5. When those two links failed traffic had to be rerouted resulting in heavier load on other links, which caused the increased delay.

Table 5.3 shows the calculated average delay for some selected cases when two failures at a time occur.

Two Failure States		
Failed Link	Failed Link	Average delay in seconds
1-2	2-3	0.1111
1-2	2-6	0.1736
1-2	3-5	0.2592
1-2	5-8	0.1675
1-2	7-8	0.1736
1-4	2-3	0.1111
1-4	2-6	0.1736
1-4	3-5	0.2592
1-4	4-5	0.2497
1-4	5-8	0.1675
2-3	4-5	0.2704
2-3	9-10	0.1874



Two Failure States		
Failed Link	Failed Link	Average delay in seconds
2-6	6-7	0.2004
3-5	6-7	0.2979
3-5	7-9	0.2798
3-5	8-9	0.3068
3-5	9-10	0.2849
3-7	9-10	0.1874
4-5	5-8	0.1864
4-5	6-7	0.2963
4-5	6-10	0.2782
4-5	8-9	0.3052
6-7	7-8	0.2004
6-10	7-8	0.1823
6-10	9-10	0.0556*
7-8	8-9	0.1222
6-7	6-10	0.0808**
7-8	7-9	0.1823

\* Path does not exist

\*\* Overflow on link 5-8

**Table 5.3** Two failures at a time delay.

As can be seen from the above table again, the average network delay increased as states were considered where link 3-5 or link 4-5 failed.

Another observation is that, as link 6-10 and 9-10 both failed, node 10 became disconnected resulting in less amount of traffic entering the network, i.e., the amount of traffic was reduced by  $\gamma_{10,4}=0.16\gamma=16.6\text{Kbps}$ .

This resulted in reduced link flows and thus a decreased network average delay.

Another thing to be observed is the failure of links 6-7 and 6-10. When both links failed an overload condition occurred on link 5-8, i.e.,  $\mu C_i < \lambda_i$  but since our routing algorithm selects secondary paths (second shortest paths), whenever an overload condition occurs, the operation of the network was not disrupted, and the condition of having an infinite network average delay was avoided. Instead, even a better performance can be observed as the network average delay went down to 0.0808 seconds.

The conclusion made from this observation is that the shortest path is not always the optimal one. When a shortest path is calculated, all traffic tends to use this path resulting in increased loading conditions of that path, and thus increased average delay.

In optimal routing network flows are calculated as to give a minimal network average delay for the network (Tanenbaum,1996). However, this approach is applicable only to certain limited conditions, and in the case of multipath routing, the situation would become more complicated since there would be an infinite number of ways to route traffic between any pair of nodes.

As stated earlier we could generate bounds on the performance measure, which is delay, using Eq. 3.20,

$$T_L(m) = \sum_{k=1}^m P(S_k)T(S_k) + \left(1 - \sum_{k=1}^m P(S_k)\right)T(S_0)$$

$$T_U(m) = \sum_{k=1}^m P(S_k)T(S_k) + \left(1 - \sum_{k=1}^m P(S_k)\right)T(S_{2^n})$$

The lower bound of the expected value of the network average delay,  $T_L(m)$  according to our algorithm:

$T_L(m)$  for no, single, and two failure states=0.3345 seconds.

The upper bound of the expected value of the network average delay is infinity, because failure corresponds to total loss of the link and  $\bar{T}$  is always infinite.

### 5.3.1 Reliability Measure

Using the estimator  $\tilde{T}_m$  of Eq. 3.21 our reliability measure, which is delay averaged over the no, single failure, and two failure states is

$$\tilde{T}_m = \frac{\sum_{k=1}^m P(S_k)T(S_k)}{\sum_{k=1}^m P(S_k)} = 0.1849 \text{ seconds}$$

This is a good estimator since 99% of all the states were covered, and  $\tilde{T}_m$  represents an average of the network delay for those states.

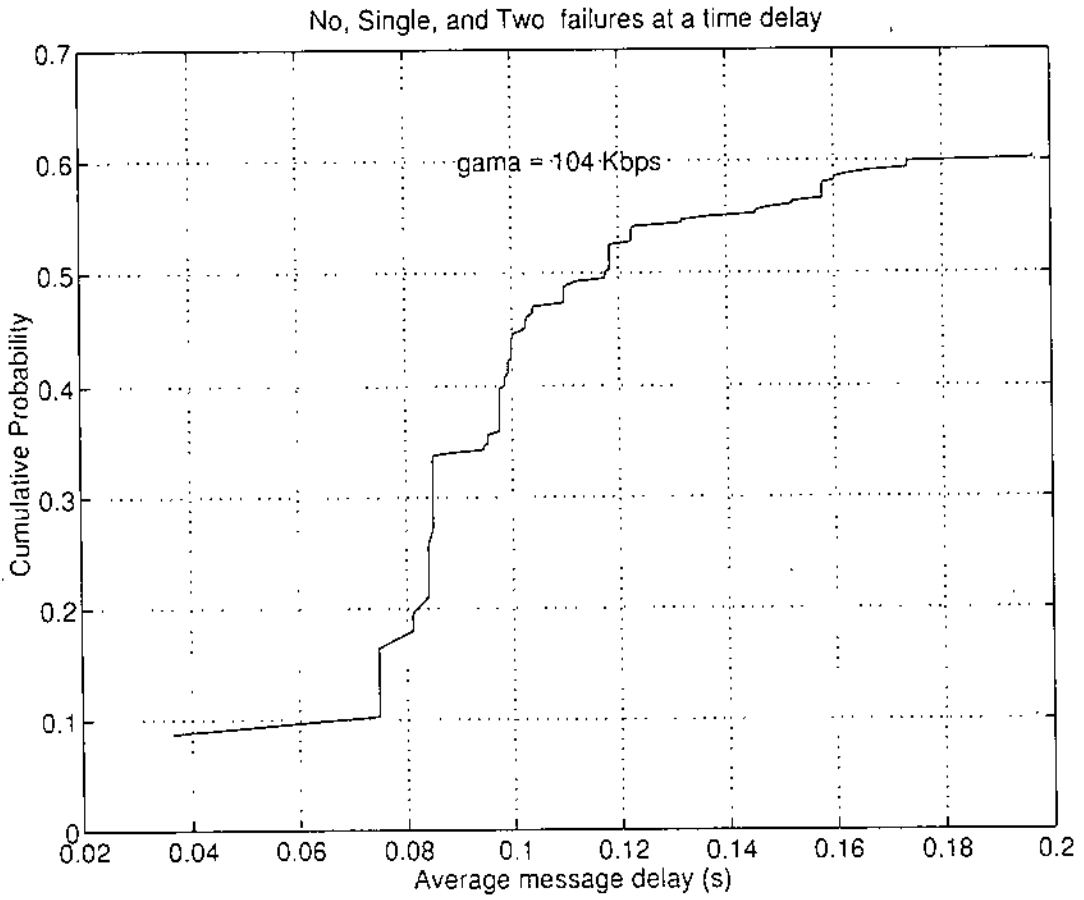
## 5.4 Effect of Availability

Our weight or shortest path measure (length) used is availability, or the probability of a link being operational.

In the network of Fig. 5.1 the availability of each link was shown next to it. As can be seen this availability is relatively high and ranges from 0.8 up to 0.999.

First of all the effect of availability on the number of states considered will be studied. As shown in Figs 5.2, 5.3, 5.4, and 5.5, the number of states considered was enough to cover a large portion of the state space. Also the sum of probabilities of occurrences of the states added up to nearly one.

The effect of decreasing the availability of the links in Fig. 5.1 can be shown in Fig. 5.6, where an availability of 0.85 for each of the links was used, and the probability distribution function plotted using Eq. 3.21.



**Fig. 5.6 Probability distribution function for decreased availability (0.85).**

As can be seen the portion of the state space covered is decreased, which means that more states have to be considered to get more accurate results concerning our performance measure. The same results for decreased availability are given in Table 5.4

State	Probability
No failure ( $S_0$ )	0.0874
Single Failures	0.2312
Two link failures	0.2857
No, Single, Two failures	0.6042

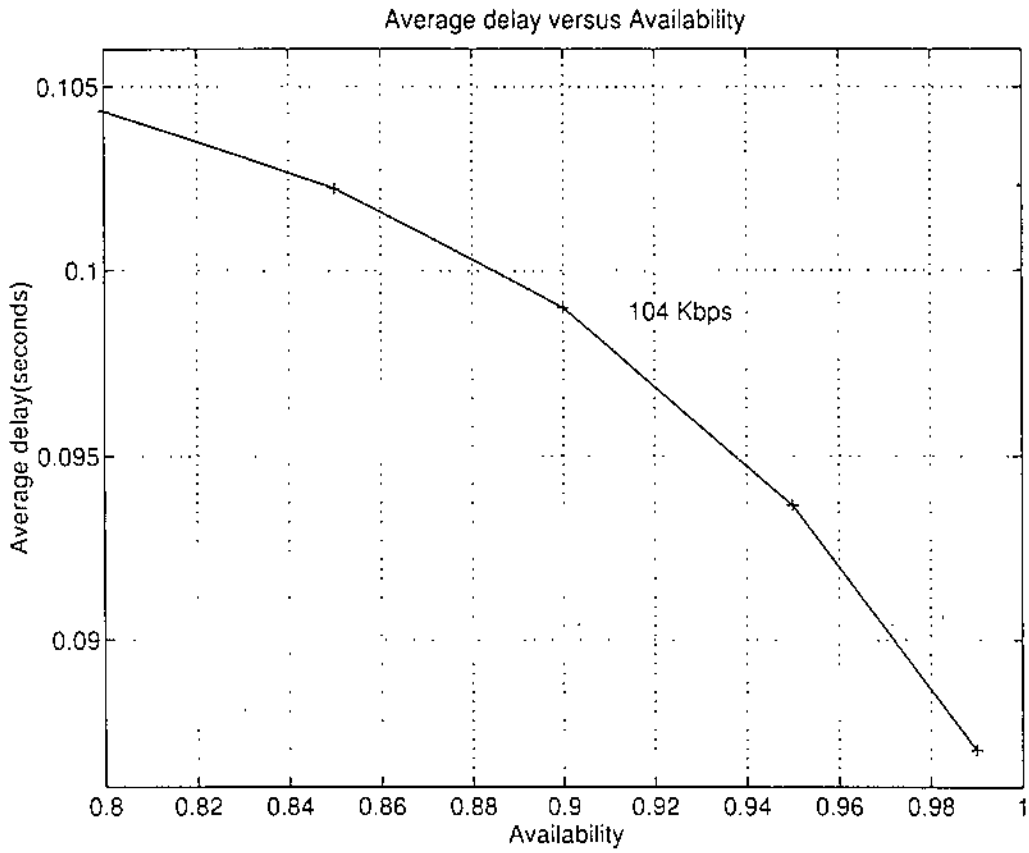
**Table 5.4** Probability of states for decreased availability (0.85).

This means only 60% of the total number of states was covered.

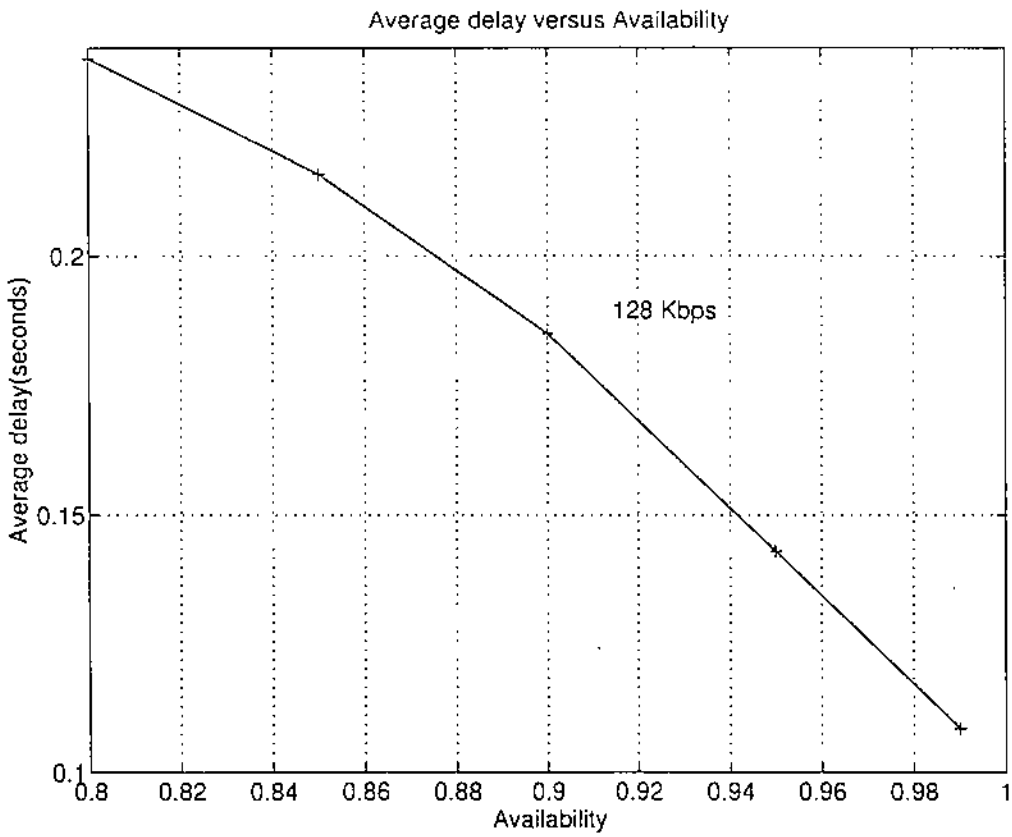
Another study is the effect of availability on network average delay, taken over the no failure, single failure, and two failure states, for different availabilities, the average network delay for those states was calculated using Eq. 3.21,

$$T_m = \frac{\sum_{k=1}^m P(S_k)T(S_k)}{\sum_{k=1}^m P(S_k)}$$

The results for Fig. 5.1 are plotted in Figs 5.7 and 5.8 for a network load of  $\gamma=104$ Kbps and  $\gamma=128$  Kbps.



**Fig. 5.7 Average delay  $\gamma=104$  Kbps.**



**Fig. 5.8 Average delay  $\gamma=128$  Kbps.**

Eq. 3.21 gives an estimator for the network average delay. This performance measure is only useful when the state changes are occurring rapidly for a network to operate on the delay curve. Also the network has to be connected in all states, otherwise infinite delay would occur.

The Figs show that the average delay decreases with increased availability. Also increasing the network load results in a higher average delay.

### 5.5 Calculation Time

The calculation time of our algorithm was tested for different loads, and different network sizes. The machine used in these tests was a 486 DX personal computer. The results of applying different loading conditions to the network of Fig. 5.1 are shown in Fig. 5.9.

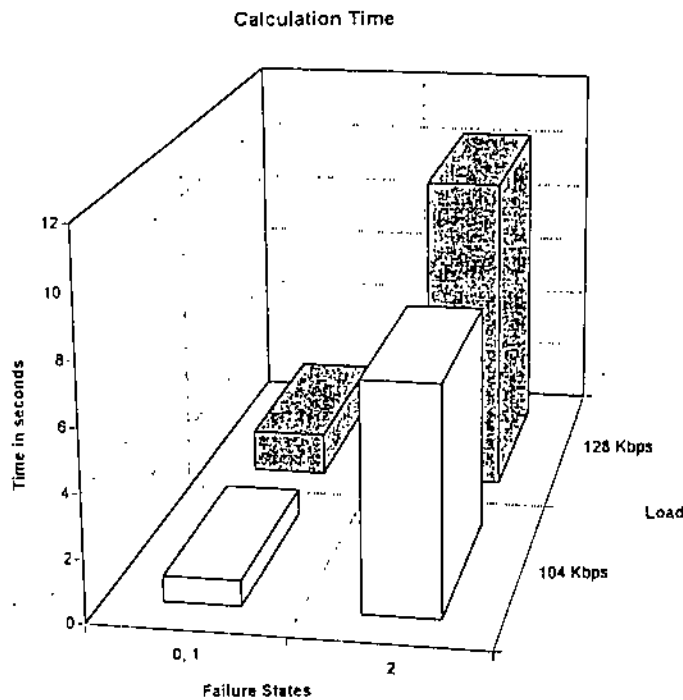
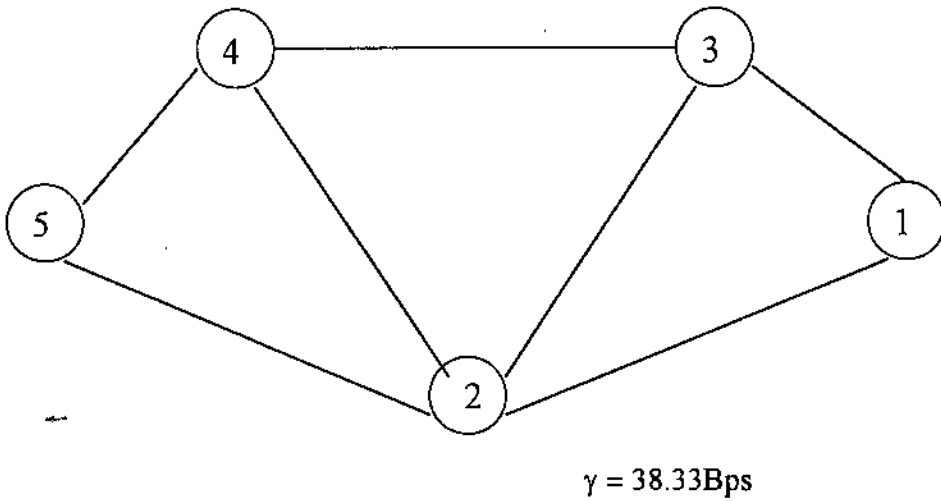


Fig. 5.9 Calculation time.



As can be seen the increased load results in an increase in calculation time. This is due to overload conditions on links, and the use of secondary routes.

Another thing which affects the calculation time is the size of the network. As an example of a small network, the following 5 node, 7 link network in Fig. 5.10 was considered (Cravis,1981).



**Fig. 5.10 A 5 node, 7 link network.**

The calculation time for this network and the 10 node 15 link network of Fig. 5.1 is shown in Fig. 5.11.

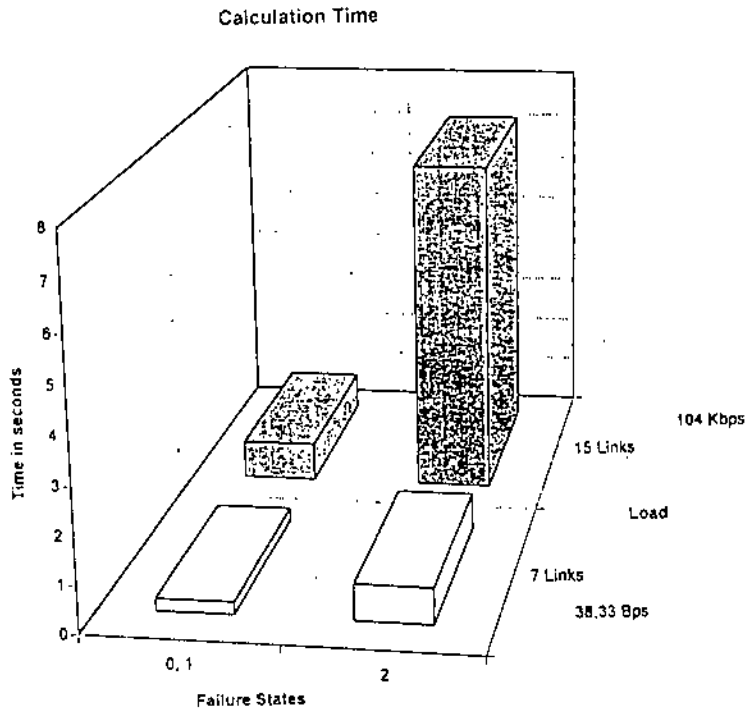
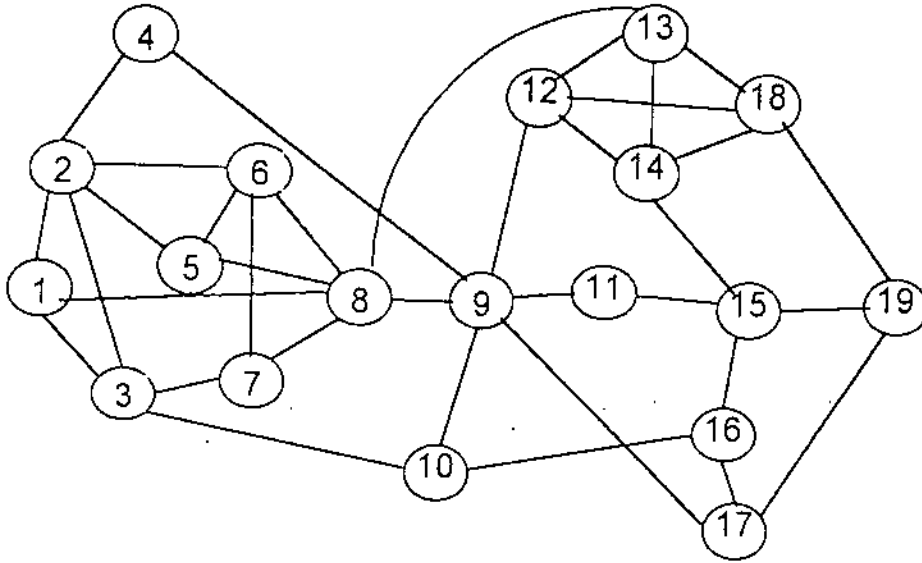


Fig. 5.11 Calculation time.

As can be seen there is a large increase in calculation time as the number of links increases, especially for the two failure at a time cases, as more states have to be considered to obtain our reliability measure. For a network of  $L$  links, the single failure states add up to  $L$ , while the two failure states add up to  $\frac{L(L-1)}{2}$ .

As a final example consider the 19 node, and 36 link network shown in Fig. 5.12 (Thaker et al,1986).



**Fig. 5.12 A 19 node, 36 link network.**

= This is a large network, requiring a total of  $1+36+\frac{36*35}{2} = 667$  cases to be evaluated. That is, for each case the shortest, and second shortest path have to be evaluated. Routing and link flows have to be calculated, and finally the average network delay for each case is computed.

The calculation time needed to solve this network was as follows:

The no, and single failure cases took 14.46 seconds.

The two failure cases took 579.02 seconds.

Thus the total calculation time was 593.47 seconds.

From those results it can be concluded that for very large networks one might be satisfied with the no, and single failure cases to achieve reasonable results in a relatively short time. The results obtained are accurate if link availability is very high. For the network of Fig. 5.12 we have chosen an availability of 0.99 for each link, and obtained the following results shown in Table 5.5,

State	Probability
No failure ( $S_0$ )	0.6964
Single Failures	0.2533
Two link failures	0.0447
No, Single, Two failures	0.9944

**Table 5.5 Probability of states for Fig. 5.12.**

It can be seen that the probability of no, and single failure cases adds up to 0.9497, i.e., 95% of the state space in covered in a relatively short time.

### 5.6 Conclusion

We have presented a new reliability measure, based on taking into consideration the most probable failure states to occur, instead of

enumerating the whole state space, which would complicate the solution of a network. In an actual network the availability, or probability of its components remaining operational is usually high, thus if we assume that only links are apt to fail, we saw that the most probable states to occur are the network states where no link failure, one failure, or at most two link failures, as those states covered more than 99% of the state space.

Our reliability measure is a performance measure, defined as the average network delays, or the average time in the system for all packets. This performance measure is based on the ability of the network to reroute traffic in the case of failure of one, or more of its links. Since, if only a small number of links fail, -as in our case two links at most- the network usually remains connected, and can be considered as a totally new network to which the routing algorithm is applied. Flows are calculated, and the average network delay is found. This performance measure is averaged over those most probable states. The number obtained is the reliability measure of interest.

As seen in the previous sections, this approach is acceptable since the average delay in the network for the no failure state does not differ much from the average delay in the case of failure.

The routing algorithm applied calculates primary and secondary shortest paths, which serve as backup in the case of failure, and tend to

minimize the average delay experienced by packets in the system, both under normal conditions and when failures occur.

The approach we have chosen could be used in network design. Since most formal design tools assume a perfect network whose components are always functional. In reality, both the links and the nodes fail, although they are generally very reliable. Our approach in contrast to the traditional one, does not separate the availability issues from the routing decisions, and concentrates on finding the most reliable path to route traffic, and continue to offer the network users the expected level of service in the cases of failure.

For further work, we have assumed independence of link failures, our method could be developed to take into account the dependence of link failures.

## References

Andrew S. Tanenbaum. 1996. Computer Networks, Third edition, Prentice-Hall, Inc. , New York.

Bezal Gavish, Irina Neuman. 1992. Routing in a Network with Unreliable Components, IEEE Transactions on Communications, Vol. 40, No.7: 1248-1258.

Brunilde Sanso, Francois , and Michel Gendreau. 1991. On the evaluation of Telecommunications Network Reliability Using Routing Models, IEEE Transactions on communications, Vol 39, No.10: 1494-1501.

Dimitri Bertsekas, Robert Gallager. 1987. Data Networks, Prentice-Hall International Editions, New York.

E. W. Dijkstra. 1959. A note on two problems in connection with graphs, Numer. Math., Vol. 1: 269-271.

Fred Halsall. 1995. Data Communications, Computer Networks and Open Systems, Fourth Edition, Addison-Wesley, Inc., England.

FUJITSU LIMITED , Personal Computer Network System Design, Second Edition, March 1994, Japan.

Gautam H. Thaker, and J. Bibb Cain. 1986. Interactions Between Routing and Flow Control Algorithms, IEEE Transactions on Communications, Vol. COM-34. No.3: 269-277.

Gerd E. Keiser. 1989. Local Area Networks, McGraw-Hill, Inc. , New York.

Howard Cravis. 1981. Communications Network Analysis, Lexington Books, Toronto.

J. M. McQuillan, Ira Richer, Eric C. Rosen. 1980. The New Routing Algorithm for the ARPANET, IEEE Transactions on Communication, Vol. COM28, No.5: 711-719.

Jean Walrand. 1991. Communication Networks, Richard D. Irwin, Inc., and Aksen Associates, Inc., Boston.



Jin Y. Yen. 1971. Finding the K Shortest Loopless Paths in a Network, Man. Sci., Vol 17: 712-716.

K.K. Aggarwal, Suresh Rai. 1981. Reliability Evaluation in Computer-Communication Networks, IEEE Transactions on Reliability, Vol. R-30, No.1: 32-35.

L. Kleinrock. 1976. Queueing systems, Volume II :Computer Applications, New York: Wiley, New York.

McQuillan, John M. Richer, Ira Rosen, Eric C. 1995. Overview of the new routing algorithm for the ARPANET, Computer Communication Review, Vol 25, No.1: 54-60.

Michael O. Ball. 1979. Computing Network Reliability, Operations Research, Vol. 27, No.4: 823-838.

Misha Schwartz, Tomas E. Stern. 1980. Routing Techniques Used in Computer Communication Networks, IEEE Transactions on Communications, Vol. COM-28, No.4: 539-552.

Mohammed J. Sutari. 1996. Communication Network Reliability Based on Lost Call Traffic, Ms.C. Thesis, University of Jordan, Amman, Jordan.

Stuart E. Dreyfus. 1969. An Appraisal of Some Shortest-Path Algorithms, J. Operations Research, Vol 17, No.3: 395-412.

Victor O.K. Li, John A. Silvester. 1984. Performance Analysis of Networks with Unreliable Components, IEEE Transactions on Communications, Vol COM-32, No.10: 1105-1110.

## **APPENDIX A1**

### **Reliability Algorithm**

%Generating the no and single failure cases

```
tim=cputime
TAVG=[];TAVG1=[];
clc;
conn;
kkk=1;
fid2=fopen('output.m','w');
inn;          %No failure case
dij3;
fprintf('\n\nCase %d (No failure) Delay T %2.5d ',kkk,TT);
fprintf(fid2,'\n\nCase %d (No failure) Delay T %2.5d ',kkk,TT);
TAVG1=[TAVG1 TT];
kkk=kkk+1;
for iii=1:n,          %Single failure cases
    for j2=iii+1:n,
        if A(iii,j2)==1
            inn;
            S(iii,j2)=inf;
            S(j2,iii)=inf;
            dij3;
            fprintf('\n\nCase %d Link %d %d Delay T %2.5d ',kkk,iii,j2,TT);
            fprintf(fid2,'\n\nCase %d Link %d %d Delay T %2.5d
',kkk,iii,j2,TT);
            kkk=kkk+1;
            TAVG1=[TAVG1 TT];
        end
    end
end
fclose(fid2);
tim1=(cputime-tim)/60

regl;
```

%Generating the two failure cases

```
tim=cputime;
TAVG2=[];
clc;
conn;
kkk=1;
fid1=fopen('output1.m','w');
for iii=1:n,          %Two failures at a time
    for j1=iii+1:n,
        if A(iii,j1)==1
            inn;
            S(iii,j1)=inf;
            S(j1,iii)=inf;
            for l=iii+1:n,
                for ll=l+1:n,
                    if A(l,ll)==1
                        S(l,ll)=inf;
                        S(ll,l)=inf;
                    end
                    dij3;
                    fprintf('\n\n Case %d link %d %d & %d %d Time
%2.4f,kkk,iii,j1,l,ll,TT);
                    fprintf(fid1,'\n\n Case %d link %d %d & %d %d Time
%2.4f,kkk,iii,j1,l,ll,TT);
                    kkk=kkk+1;
                    TAVG2=[TAVG2 TT];
                end
            end
        end
    end
end
for iii=1:n,          %Two failures at a time
    for j1=iii+1:n,
        if A(iii,j1)==1
            inn;
            S(iii,j1)=inf;
            S(j1,iii)=inf;
            for l=j1+1:n,
```

```
if A(iii,l)==1
    S(iii,l)=inf;
    S(l,iii)=inf;
    dij3;
    fprintf('\n\n Case %d link %d %d & %d %d Time
%2.4f,kkk,iii,j1,iii,l,TT);
    fprintf(fid1,'\n\n Case %d link %d %d & %d %d Time
%2.4f,kkk,iii,j1,iii,l,TT);
    TAVG2=[TAVG2 TT];
    kkk=kkk+1;
    inn;
    S(iii,j1)=inf;
    S(j1,iii)=inf;
end
end
end
end
end
fclose(fid1);
tim2=(cputime-tim)/60
```

```
%Dijkstra's Algorithm for the Shortest Route  
%Calculating the first shortest route(Primary Path  
%Calculating the second shortest route(Secondary
```

```
R=[zeros(n)];RR=[zeros(n)];la=[zeros(n)];  
for ii=1:n,  
    for jj=1:n,  
        TL=[];VECT=zeros([1,n]);PP=[]; %Initialization  
        nn=n;s=0;t=0;i=0;p=0;tt=0;t1=0;  
        if ga(ii,jj)~=0  
            s=ii; %Reading input values  
            t=jj;  
            t1=t;  
            for j=1:nn, %Initialization  
                TL(j)=inf;  
            end  
            TL(s)=0;  
            VECT(s)=1;  
            i=s;  
            while p~=t %Are all nodes processed  
                m=inf;  
                for j=1:nn,  
                    if VECT(j)==0  
                        TL(j)=min(TL(j),(TL(i)+S(i,j))); %assignment  
                        if TL(j)<=m  
                            m=TL(j);  
                            p=j;  
                        end  
                    end  
                end  
                VECT(p)=1; %Permanent label  
                i=p;  
            end  
            if TL(t)==inf  
                % fprintf('\n\n Case %d Primary Path does not exist  
                does not exist  
                %fprintf(fid1,'\n\n Case %d Path does not exist  
                end  
                % fprintf('\n Primary Path %f,TL(t));  
                tt=t;  
                VECT=zeros([1,n]); %The size of  
                VECT(t)=1;
```

```
if TL(t1)~=inf
    while t~=s
        for j=1:nn,
            if VECT(j)==0
                if TL(j)+S(j,t)==TL(t)
                    PP=[PP,j];           %The Adjecant Node Matrix
                    t=j;
                end
            end
        end
        end
    end
    PP=[t1,PP];
    i=1;
    while i~=length(PP),                %The Successor Node Matrix
        R(PP(i),s)=PP(i+1);
        i=i+1;
    end
    if TL(t1)~=inf
        no1=0;no2=0;ss1=[];             %Secondary Path
        k=length(PP);
        while k~=1
            no1=PP(k);
            no2=PP(k-1);
            ss1=[ss1,S(no1,no2)];
            S(no1,no2)=inf;
            S(no2,no1)=inf;
            k=k-1;
        end
    end

    TLL=[];VECTT=zeros([1,n]);PPP=[];  %Initialization
    nn=n;i=0;p=0;t1=0;

    for j=1:nn,                          %Initialization
        TLL(j)=inf;
    end
    TLL(s)=0;
    VECTT(s)=1;
    i=s;
    while p~=t1                          %Are all nodes permanently labeled
        m=inf;
        for j=1:nn,
            if VECTT(j)==0
```



```

        TLL(j)=min(TLL(j),(TLL(i)+S(i,j))); %Assigning temporality
labels
    if TLL(j)<=m
        m=TLL(j);
        p=j;
    end
end
end
VECTT(p)=1; %Permanent Label
i=p;
end
if TLL(t1)==inf
    %fprintf('\n\n Case %d Secondary Path does not exist
',kkk);%Path does not exist
    %fprintf(fid1,'\n\n Case %d Path does not exist ',kkk);
end
    % fprintf('\n Secondary Path %f\n\n',TLL(tt)); %The
shortest distance
    tt=tt;
    VECTT=zeros([1,n]); %The shortest path
    VECTT(t1)=1;
    if TLL(t1)~=inf
        while tt~=s
            for j=1:nn,
                if VECTT(j)==0
                    if TLL(j)+S(j,tt)==TLL(tt)
                        PPP=[PPP,j]; %The Adjecant Node Matrix
                        tt=j;
                    end
                end
            end
        end
    end
    PPP=[t1,PPP];
    i=1;
    while i~=length(PPP), %The Successor Node Matrix
        RR(PPP(i),s)=PPP(i+1);
        i=i+1;
    end
end
end
k=length(PP);
k1=1;
while k~=1
    no1=PP(k);

```

```
no2=PP(k-1);  
S(no1,no2)=ss1(k1);  
S(no2,no1)=ss1(k1);  
k=k-1;  
k1=k1+1;  
end  
end  
end  
end  
flows;  
delay;  
rrr;
```

```

%Link flows
%Calculating the flow in each link

for i=1:n,
for j=1:n,
if ga(i,j)~=0      %Traffic generated at i destined for j
k=i;
no=R(k,j);
if no~=0
while k~=j      %Primary Path (while)
no=R(k,j);
la(k,no)=la(k,no)+ga(i,j);
if la(k,no)>=(C(k,no)*u)      %Overflow using primary path
fprintf('\n Case %d Primary Path Overflow link %d %d
',kkk,k,no);
k=i;
while k~=j
no=R(k,j);
la(k,no)=0;
k=no;
end
k=i;
no=RR(k,j);
if no~=0
while k~=j      %2'nd while loop
no=RR(k,j);
la(k,no)=la(k,no)+ga(i,j);
if la(k,no)>=(C(k,no)*u)
fprintf('\n Case %d Secondary Path Overflow link %d
%d,kkk,k,no');
la(k,no)=(C(k,no)*u);
end
k=no;
end %2'nd while loop
end
if no==0 %If Secondary Path doesn't exist
k=i;
while k~=j
no=R(k,j);
la(k,no)=la(k,no)+ga(i,j);
if la(k,no)>=(C(k,no)*u)
la(k,no)=(C(i,j)*u);

```

```
        end
        k=no;
    end
    end
end    %Overflow using primary path
k=no;
end    %Primary Path (while)
end
end    %If ga(i,j)~=0
end
end
```

% Average Network Delay

%Calculating the delay of all messages in the network

```
T=0;TT=0;den=0;
for i=1:n,
for j=1:n,
    if i~=j
        den=(u*C(i,j)-la(i,j));
        if den<=0
            fprintf('\n\n Case %d Infinite delay Overflow on link %d %d',kkk,i,j);
%    fprintf(fid1,'\n\n Case %d Infinite delay Overflow on link %d
%d',kkk,i,j);
            T=T+inf;
        else
            T=T+(la(i,j)/den);
        end
    end
end
end
TT=T/gaa;
TAVG=[TAVG TT];          %Delay for all cases
```

%Output File

```
fid=fopen('output3.m','w');
fprintf(fid,'\n\n Successor Node Matrix (Primary route)\n');
for i=1:n,
    for j=1:n,
        fprintf(fid,' %d',R(i,j));
    end
    fprintf(fid,'\n');
end

fprintf(fid,'\n\n Successor Node Matrix (Secondary route)\n');
for i=1:n,
    for j=1:n,
        fprintf(fid,' %d',RR(i,j));
    end
    fprintf(fid,'\n');
end
```

```
fprintf(fid,'\n\n Link Flows\n');  
for i=1:n,  
    for j=1:n,  
        fprintf(fid,' %3.2f    ',la(i,j));  
    end  
    fprintf(fid,'\n');  
end  
  
fprintf(fid,'\n\n Network Average Delay\n');  
fprintf(fid,'T Delay %d ',TT);  
fclose(fid);
```

## **APPENDIX A2**

**Calculating the Probability of the States**

**Generating the Probability Distribution Function**

**Calculating the Average Delay**

%Calculating the probability of no and single failures  
%Calculating the Cumulative probability

```
tim=cputime;
clc;clg;
conn;PSK=[];RPQ=[];CP=[];inn;SPSK=0;TPSK=0;TLM=0;TUM=0;
fid2=fopen('output2.m','w');
for i=1:n,
    for j=i+1:n,
        if A(i,j)==1
            RPQ(i,j)=1/(10^S(i,j));
        end
    end
end
state2;
kkk=0;
fprintf('\n\nCase %d (No failure) Probability P %e',kkk,pps);
fprintf(fid2,'\n\nCase %d (No failure) Probability P %e',kkk,pps);
kkk=1;
for iii=1:n,
    for j2=iii+1:n,
        if A(iii,j2)==1
            inn;
            S(iii,j2)=inf;
            S(j2,iii)=inf;
            state2;
            fprintf('\n\nCase %d Link %d %d Probability P %e',kkk,iii,j2,pps);
            fprintf(fid2,'\n\nCase %d Link %d %d Probability P %e',kkk,iii,j2,pps);
            kkk=kkk+1;
        end
    end
end

for i=1:length(PSK) %Calculating the Cumulative probability
    SPSK=SPSK+PSK(i);
    TPSK=TPSK+(PSK(i)*TAVG(i));
    CP=[CP SPSK];
end
```



```
(cputime-tim)/60
```

```
fprintf('\n\nProbability of No and Single failure Cases = %e',SPSK);  
fprintf(fid2,'\n\nProbability of No and Single failure Cases Cases =  
%e',SPSK);
```

```
figure(1)  
plot(sort(TAVG1),CP)  
title('Single and no failure delay'),...  
xlabel('Average message delay (s)'),...  
ylabel('Cumulative Probability'),grid  
text(.23,.93,'gama = 104 Kbps ')
```

```
fclose(fid2);  
reg6;
```

```
%Calculating the probability of two failures at a time  
%Calculating the Cumulative probability
```

```
tim=cputime;  
clc;  
conn;PSK=[];RPQ=[];CPP=[];inn;  
kkk=1;  
fid1=fopen('output6.m','w');  
for i=1:n,  
    for j=i+1:n,  
        if A(i,j)==1  
            RPQ(i,j)=1/(10^S(i,j));  
        end  
    end  
end  
for iii=1:n,  
    for j1=iii+1:n,  
        if A(iii,j1)==1  
            inn;  
            S(iii,j1)=inf;  
            S(j1,iii)=inf;  
            for l=iii+1:n,  
                for ll=l+1:n,  
                    if A(l,ll)==1
```

```
S(l,ll)=inf;
  S(ll,l)=inf;
  state2;
  fprintf('\n\n Case %d link %d %d & %d %d Probability
%e',kkk,iii,jl,l,ll,pps);
  fprintf(fid1,'\n\n Case %d link %d %d & %d %d Probability
%e',kkk,iii,jl,l,ll,pps);
  kkk=kkk+1;
  inn;
  S(iii,jl)=inf;
  S(jl,iii)=inf;
end
end
end
end
end
for iii=1:n,
  for jl=iii+1:n,
    if A(iii,jl)==1
      inn;
      S(iii,jl)=inf;
      S(jl,iii)=inf;
      for l=jl+1:n,
        if A(iii,l)==1
          S(iii,l)=inf;
          S(l,iii)=inf;
          state2;
          fprintf('\n\n Case %d link %d %d & %d %d Probability
%e',kkk,iii,jl,iii,l,pps);
          fprintf(fid1,'\n\n Case %d link %d %d & %d %d Probability
%e',kkk,iii,jl,iii,l,pps);
          kkk=kkk+1;
          inn;
          S(iii,jl)=inf;
          S(jl,iii)=inf;
        end
      end
    end
  end
end
end
end
for i=1:length(PSK)
```

```
SPSK=SPSK+PSK(i);
TPSK=TPSK+(PSK(i)*TAVG2(i));
CP=[CP SPSK];    %Cumulative probability of no single and two
failures
CPP=[CPP SPSK];    %Cumulative probability of two failures
end
TPSK=TPSK/SPSK;
TLM=TPSK+(1-TPSK)*TAVG1(1);
TUM=TPSK+(1-TPSK)*inf;

(cputime-tim)/60

fprintf('\n\nLower bound on expected average delay %e ',TLM);
fprintf('\nUpper bound on expected average delay %e ',TUM);
fprintf(fid1,'\n\nLower bound on expected average delay %e ',TLM);
fprintf(fid1,'\nUpper bound on expected average delay %e ',TUM);
fprintf('\n\nProbability of no single and two failure Cases = %e',SPSK);
fprintf(fid1,'\n\nProbability of no single and two failure Cases =
%e',SPSK);
fprintf('\n\nAverage delay for all cases = %e',TPSK);
fprintf(fid1,'\n\nAverage delay for all cases = %e',TPSK);

figure(2)
plot(sort(TAVG2),CPP),
title('Two failures at a time delay'),...
xlabel('Average message delay (s)'),...
ylabel('Cumulative Probability'),grid
text(.23,.995,'gama = 104 Kbps');

figure(3)
plot(sort(TAVG),CP),
title('No Single and Two failures at a time delay'),...
xlabel('Average message delay (s)'),...
ylabel('Cumulative Probability'),grid
text(.23,.93,'gama = 104 Kbps');

fclose(fid1);
```

%Calculating the probability of a network state

```
pps=1;RRRQ=0;
for i=1:n,
    for j=i+1:n,
        if A(i,j)==1
            if S(i,j)~=inf
                RRRQ=1;
            else
                RRRQ=(1-RPQ(i,j))/RPQ(i,j);
            end
            pps=pps*RPQ(i,j)*(RRRQ);
        end
    end
end
PSK=[PSK pps];
```

## **APPENDIX A3**

### **Plotting Results**

**Average Time versus Availability**

**Calculation Time**

**Probability Distribution Function**

```
load amina.mat;load aminal.mat;load timmm.mat
AMINA=[T0 T1 T2 T3 T4];UX=[.99 .95 .9 .85 .8];
AMINA1=[TT0 TT1 TT2 TT3 TT4];hh1=[tim1 tim2];hh2=[timm1
timm2];
cas=[1 2];
clg
```

```
figure(1)
plot(UX,AMINA,'y-',UX,AMINA,'y+',UX,AMINA1,'y-
',UX,AMINA1,'y+'),
title('Average delay versus Availability'),...
xlabel('Availability'),...
ylabel('Average delay'),grid
gtext('104 Kbps');
gtext('128 Kbps');
```

```
figure(2)
plot(UX,AMINA,'y-',UX,AMINA,'y+'),
title('Average delay versus Availability'),...
xlabel('Availability'),...
ylabel('Average delay'),grid
gtext('104 Kbps');
```

```
figure(3)
plot(UX,AMINA1,'y-',UX,AMINA1,'y+'),
title('Average delay versus Availability'),...
xlabel('Availability'),...
ylabel('Average delay'),grid
gtext('128 Kbps');
```

```
figure(4)
plot(cas,hh1,'y-',cas,hh1,'y+',cas,hh2,'y-',cas,hh2,'y+'),
title('Calculation Time'),...
xlabel('Cases'),...
ylabel('Time in seconds'),grid
gtext('104 Kbps');
gtext('128 Kbps');
gtext('15 Link Network');
```

```
clg
plot(sort(TTAVG),CCCP,'yo',sort(SSAVG),SSCP,'yx',sort(FFAVG),FFCP,
'y+',sort(XXAVG),XXCP,'y*',...
sort(TTAVG),CCCP,'y-',sort(SSAVG),SSCP,'y-',sort(FFAVG),FFCP,'y-
',sort(XXAVG),XXCP,'y-'),
title('No Single and Two failures at a time delay'),...
xlabel('Average message delay (s)'),...
ylabel('Cumulalative Probability'),grid
axis([0.052 0.252 0.55 .94]),
gtext('104 Kbps');
gtext('182 Kbps');
gtext('208 Kbps');
gtext('234 Kbps');
```

## **APPENDIX A4**

### **Sample of Input & Output File for Fig. 5.1**

**Connection Matrix**

**Successor Node Matrix**

**Shortest Route Weight Matrix**

**Link Flows**

**Average Time in the System**



%Input file

%Connection matrix

```
n=10;  
A=[0 1 0 1 0 0 0 0 0 0;  
  1 0 1 0 0 1 0 0 0 0;  
  0 1 0 1 1 0 1 0 0 0;  
  1 0 1 0 1 0 0 0 0 0;  
  0 0 1 1 0 0 0 1 0 0;  
  0 1 0 0 0 0 1 0 0 1;  
  0 0 1 0 0 1 0 1 1 0;  
  0 0 0 0 1 0 1 0 1 0;  
  0 0 0 0 0 0 1 1 0 1;  
  0 0 0 0 0 1 0 0 1 0];
```

%Network data

```
n=10;V=[1 2 3 4 5 6 7 8 9 10];u=1;  
S=[0 4.345e-4 inf 4.345e-4 inf inf inf inf inf;  
  4.345e-4 0 4.365e-3 inf inf 9.691e-2 inf inf inf inf;  
  inf 4.365e-3 0 2.228e-2 4.345e-4 inf 7.058e-2 inf inf inf;  
  4.345e-4 inf 2.228e-2 0 4.345e-4 inf inf inf inf inf;  
  inf inf 4.345e-4 4.345e-4 0 inf inf 4.576e-2 inf inf;  
  inf 9.691e-2 inf inf inf 0 4.345e-4 inf inf 4.345e-4;  
  inf inf 7.058e-2 inf inf 4.345e-4 0 4.345e-4 4.345e-4 inf;  
  inf inf inf inf 4.576e-2 inf 4.345e-4 0 4.345e-4 inf;  
  inf inf inf inf inf inf 4.345e-4 4.345e-4 0 4.345e-4;  
  inf inf inf inf inf 4.345e-4 inf inf 4.345e-4 0];
```

```
C=50.*[ones(n)];
```

```
gaa=104;|a=[zeros(n)];
```

```
ga=[zeros(n)];
```

```
ga(2,8)=.11*gaa;ga(8,2)=ga(2,8);ga(6,7)=.1*gaa;ga(7,6)=ga(6,7);
```

```
ga(4,10)=.16*gaa;ga(10,4)=ga(4,10);ga(3,9)=.13*gaa;ga(9,3)=ga(3,9);
```

Output File for the No Failure Case (S1)

Successor Node Matrix (Primary route)

```

0 2 0 0 0 0 0 4 0 0
0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 5 0
0 1 0 0 0 0 0 5 0 5
0 4 3 4 0 0 0 8 8 8
0 0 0 0 0 0 7 0 0 0
0 0 0 0 0 6 0 0 0 0
0 5 5 5 0 0 0 0 9 9
0 0 8 8 0 0 0 0 0 10
0 0 0 9 0 0 0 0 0 0

```

Successor Node Matrix (Secondary route)

```

0 0 0 4 0 0 0 0 0 2
0 0 0 1 0 0 0 3 0 3
0 2 0 2 0 0 0 7 7 7
0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0
0 0 0 7 0 0 10 0 0 10
0 3 3 3 0 9 0 8 9 6
0 7 0 0 0 0 0 0 0 0
0 0 7 0 0 10 7 0 0 0
0 0 0 6 0 6 9 0 0 0

```

Shortest Route Weight Matrix (S)

```

0.000000 0.000435 Inf 0.000435 Inf Inf Inf Inf Inf Inf
0.000435 0.000000 0.004365 Inf Inf 0.096910 Inf Inf Inf Inf
Inf 0.004365 0.000000 0.022280 0.000435 Inf 0.070580 Inf Inf Inf
0.000435 Inf 0.022280 0.000000 0.000435 Inf Inf Inf Inf Inf
Inf Inf 0.000435 0.000435 0.000000 Inf Inf 0.045760 Inf Inf
Inf 0.096910 Inf Inf Inf 0.000000 0.000435 Inf Inf 0.000435
Inf Inf 0.070580 Inf Inf 0.000435 0.000000 0.000435 0.000435 Inf
Inf Inf Inf Inf 0.045760 Inf 0.000435 0.000000 0.000435 Inf
Inf Inf Inf Inf Inf Inf 0.000435 0.000435 0.000000 0.000435
Inf Inf Inf Inf Inf 0.000435 Inf Inf 0.000435 0.000000

```

Link Flows

0.00	11.44	0.00	11.44	0.00	0.00	0.00	0.00	0.00	0.00
11.44	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	13.52	0.00	0.00	0.00	0.00	0.00
11.44	0.00	0.00	0.00	28.08	0.00	0.00	0.00	0.00	0.00
0.00	0.00	13.52	28.08	0.00	0.00	0.00	41.60	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	10.40	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	10.40	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	41.60	0.00	0.00	0.00	30.16	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	30.16	0.00	16.64
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	16.64	0.00

Network Average Delay

T Delay 1.822878e-001

433731

## المخلص

### اعتمادية شبكات الحاسوب

إعداد

( باربرا امينة ) محمد الإماضنة

إشراف

الاستاذ الدكتور جميل أيوب

تبحث هذه الرسالة في اعتمادية شبكات الحاسوب ذات الرزمة عند تعرضها لظروف العطل العشوائي .  
تعتمد الطرق التقليدية في الاعتمادية على حساب مقدار الربط الموجود بين نقطتين ، أو على حساب احتمالية نجاح الاتصال بين نقطتين . وتعتمد طرق أخرى على حساب كفاءة الشبكة عند خلوها من العطل . النموذج المقترح من قبلنا يأخذ بمقاييس الاعتمادية السابقة الذكر بعين الاعتبار ، ويحاول أن يكون أقرب إلى ظروف العمل الحقيقية لأي شبكة حيث أن هذا النموذج يستند إلى حقيقة واقعة وهي مقدرة الشبكة على الاستمرار في نقل المعلومات عند تعرض بعض خطوطها للعطل ، وإمكانية حساب كفاءة الشبكة تحت هذه الظروف .  
مقياس الكفاءة يعتمد على الطريقة المتبعة لتسيير الرزم في الشبكة ، وعلى الافتراض أن الرزم تغير مسارها في حال حدوث عطل في مسارها الأصلي . تتعرض الشبكات الواقعة إلى العطل بحيث يرتبط بكل خط فيها احتمالية معينة هي : احتمالية العمل الصحيح ، أو احتمالية عدم التعرض للعطل ، ويعتمد على هذه الاحتمالية وضع معين للشبكة ، بحيث عندما تكون الشبكة في وضع معين تكون بعض خطوطها وبعضها قد تعرض للعطل . في هذه

الدراسة مقياس الكفاءة الذي تبينناه هو متوسط الوقت اللازم لنقل جميع الرزم على الشبكة. حيث يحسب هذا الوقت في الأوضاع الأكثر احتمالاً التي تكون الشبكة فيها ، مثل : الأوضاع التي لا يحصل فيها عطل ، الأوضاع التي تعطل فيها خط واحد فقط ، الأوضاع التي يتعطل فيها خطين في نفس الوقت . الطرق المتبعة لنقل الرزم تعتمد على طريقة أقصر مسار حيث كلمة أقصر تعني المسار ذات الاعتمادية الأعلى ، حيث يستعمل هذا المسار في ظروف العمل الاعتيادية وثاني أقصر مسار ، حيث يستعمل عند زيادة عدد الرزم الموجهة على المسار الأول ، يأتي هذا الاتجاه في محاولة لتقليل الوقت اللازم لمرور عبر الشبكة . بعد الحصول على متوسط الوقت اللازم لنقل الرزم عبر الشبكة في جميع أوضاع الشبكة الأكثر احتمالاً يمكن الحصول على متوسط لهذه الأوقات بحيث يكون هذا المتوسط مقياس الكفاءة الذي نبحث عنه . تم تطبيق هذه الطريقة على شبكات مختلفة الحجم ، كما تم تأثير كمية الرزم ، وتأثير تغيير الاحتمالية الخاصة بعمل الخطوط على مقياس الكفاءة المتخذ .

إن هذه الطريقة توفر مقياساً مفيداً وقريباً للواقع يمكن أن يستعمل لقياس كفاءة الشبكات ، كما يمكن الاستفادة منه في تصميم شبكات الحاسوب المختلفة .