

INFORMATION TO USERS

This material was produced from a microfilm copy of the original document. While the most advanced technological means to photograph and reproduce this document have been used, the quality is heavily dependent upon the quality of the original submitted.

The following explanation of techniques is provided to help you understand markings or patterns which may appear on this reproduction.

1. The sign or "target" for pages apparently lacking from the document photographed is "Missing Page(s)". If it was possible to obtain the missing page(s) or section, they are spliced into the film along with adjacent pages. This may have necessitated cutting thru an image and duplicating adjacent pages to insure you complete continuity.
2. When an image on the film is obliterated with a large round black mark, it is an indication that the photographer suspected that the copy may have moved during exposure and thus cause a blurred image. You will find a good image of the page in the adjacent frame.
3. When a map, drawing or chart, etc., was part of the material being photographed the photographer followed a definite method in "sectioning" the material. It is customary to begin photoing at the upper left hand corner of a large sheet and to continue photoing from left to right in equal sections with a small overlap. If necessary, sectioning is continued again — beginning below the first row and continuing on until complete.
4. The majority of users indicate that the textual content is of greatest value, however, a somewhat higher quality reproduction could be made from "photographs" if essential to the understanding of the dissertation. Silver prints of "photographs" may be ordered at additional charge by writing the Order Department, giving the catalog number, title, author and specific pages you wish reproduced.
5. PLEASE NOTE: Some pages may have indistinct print. Filmed as received.

Xerox University Microfilms

300 North Zeeb Road
Ann Arbor, Michigan 48106

77-25,653

DALAL, Yogen Kantilal, 1950-
BROADCAST PROTOCOLS IN PACKET
SWITCHED COMPUTER NETWORKS.

Stanford University, Ph.D., 1977
Computer Science

Xerox University Microfilms, Ann Arbor, Michigan 48106

© 1977

by

Yogen Kantilal Dalal

BROADCAST PROTOCOLS IN PACKET SWITCHED COMPUTER NETWORKS

A DISSERTATION

SUBMITTED TO THE DEPARTMENT OF ELECTRICAL ENGINEERING

AND THE COMMITTEE ON GRADUATE STUDIES

OF STANFORD UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

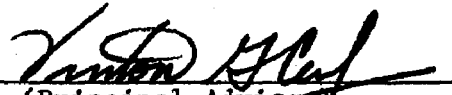
DOCTOR OF PHILOSOPHY

By

Yogen Kantilal Dalal

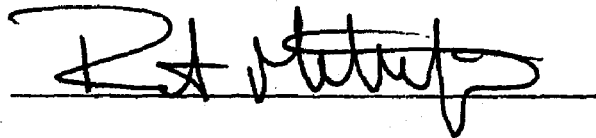
April 1977

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

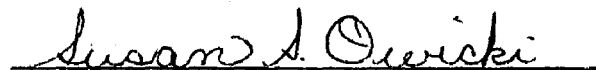


(Principal Advisor)

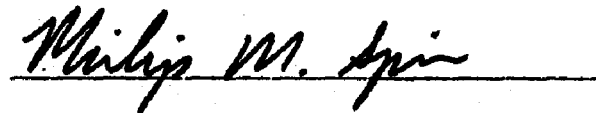
I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.




I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.



I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.



Approved for the University Committee
on Graduate Studies:



Dean of Graduate Studies

ABSTRACT

This thesis investigates the design and analysis of broadcast routing algorithms for use in store-and-forward packet switched computer networks. Broadcast routing is taken here to be a special case of multi-destination routing, in which a packet is delivered to all destinations rather than to some subset.

We examine five alternatives to transmitting separately addressed packets from the source to the destinations. The algorithms are compared qualitatively in terms of memory requirements, ease of implementation, adaptiveness to changing network conditions, and reliability. The algorithms are also compared quantitatively in terms of the number of packet copies generated to perform broadcast and the delays to propagate the packet to all destinations. Lower bounds on the performance measures are determined for all the algorithms by examining regular graphs.

Protocols that provide reliable communication using broadcast routing, i.e. broadcast protocols, are analogous to interprocess communication protocols except that communication is between one process and many processes. Reliable broadcast protocol design is faced with problems similar to those in the design of interprocess communication protocols - addressing, sequencing, duplicate detection and guarantee of delivery. This area presents many subjects for future research.

We describe a few applications for broadcast protocols in distributed computing environments. In particular, we show in detail how the catalog of a distributed file system could be structured in a simple way, if the system could make use of efficient reliable broadcast protocols.

The properties of reliable broadcast protocols at the host level emerge from the reliability of the routing algorithms and the applications for the protocols. We have examined the tradeoffs between global and subgroup broadcast routing. One conclusion we offer is that communication subnets should support both capabilities in the form of multi-destination addressing and reverse path forwarding respectively.

An outcome of the investigation of broadcast routing algorithms is the formulation of two distributed (parallel) algorithms for constructing minimal spanning trees. We believe that these algorithms are the first of their kind. The formulation of such algorithms has made the problems affecting the design of distributed algorithms in network environments clearer. These minimal spanning tree algorithms can be used in broadcast routing, as well as other networks like the Packet Radio Network.

ACKNOWLEDGEMENTS

I wish to thank my advisor, Vinton G. Cerf, for many things: for his excellent guidance and teaching, for his great sense of humor, his good counsel and advice in all matters, and the opportunity to do so much as a graduate student. He has made my graduate study a very worthwhile and enjoyable experience. I also thank Robert M. Metcalfe for his interest in my work, many helpful discussions and ideas, and for the excellent job he did as my reader. I thank Philip M. Spira and Robert E. Tarjan for many valuable discussions, suggestions and comments that have helped this research. I thank Susan Owicki for her helpful comments as my reader.

I thank my friends and colleagues, in particular B. Ramakrishna Rau and Carl Sunshine for their willingness to discuss many a wild idea, and their criticism which helped crystallize the better ones. My fellow students at the Digital Systems Lab, Dick Karp, Ron Crane, Jim Mathis, Judy Estrin and Darryl Rubin have helped in many ways. Carolyn Taynai is one of the nicest people I have met, and did a great deal to make sure that things went smoothly.

My research has been supported by the Advanced Research Projects Agency of the Department of Defense, under ARPA Order No. 2494, Contract No. MDA903-76C-0093, and the National Science Foundation, under research grant MCS73-07973-A1,2. Facilities and individuals at several

ARPA sponsored institutions including the Digital Systems Lab and the SUMEX-AIM project at Stanford University, the Information Sciences Institute of USC, and Bolt Beranek and Newman Inc. have been instrumental in producing this report. The computation facilities at the Stanford Linear Accelerator Center (SLAC) were used to produce graphs for some of my figures.

My thanks go to my parents and family for their constant encouragement and support all these years, and for making graduate study at Stanford possible in the first place.

TABLE OF CONTENTS

<u>Subject</u>	<u>Page</u>
ABSTRACT	iv
ACKNOWLEDGEMENTS	vi
TABLE OF CONTENTS	viii
LIST OF TABLES	xi
LIST OF FIGURES	xii
CHAPTER	
1 INTRODUCTION	1
1.1 Introduction	1
1.2 Summary	7
2 DISTRIBUTED MINIMAL SPANNING TREE ALGORITHMS	9
2.1 Introduction	9
2.2 Construction Principles for Minimal Spanning Trees . .	13
2.3 Parallel MST algorithms	17
2.4 A Static Distributed MST Algorithm	21
2.5 An Alternate Model	43
2.6 An Adaptive Distributed MST Algorithm	50
2.7 Conclusions	78
3 BROADCAST ROUTING ALGORITHMS	79
3.1 Introduction	79
3.2 Separately Addressed Packets	81

Table of Contents

3.3	Multi-Destination Addressing	84
3.4	Hot Potato Forwarding	88
3.5	Source Based Forwarding	91
3.6	Reverse Path Forwarding	93
3.7	Forwarding along a Spanning Tree	102
3.8	Reliability of Broadcast Routing Protocols	105
3.9	Global vs Subgroup Broadcast Routing Algorithms	108
3.10	Conclusions	110
4	PERFORMANCE EVALUATION OF BROADCAST ROUTING ALGORITHMS	114
4.1	Introduction	114
4.2	Performance Measures	116
4.3	Regular Graphs	118
4.4	Separately Addressed Packets (SAP)	129
4.5	Multi-Destination Addressing (MDA) and Source Based Forwarding (SBF)	139
4.6	Hot Potato Forwarding	142
4.7	Reverse Path Forwarding	146
4.8	Minimal Spanning Tree Forwarding	149
4.9	Comparison of the Different Schemes	154
4.10	Performance in the ARPANET	157
4.11	Conclusions	162
5	DISTRIBUTED FILE SYSTEMS: AN APPLICATION	163
5.1	Introduction	163
5.2	User Interface and Catalog Structure	169

Table of Contents

5.3	File Migration	198
5.4	Multiple Copies of a File	201
5.5	Conclusions	205
6	CONCLUSIONS AND FUTURE RESEARCH	207
APPENDIX A	DETAILED TABLE OF CONTENTS	212
APPENDIX B	AN EXAMPLE OF REDUNDANT COMPUTATION OWING TO COMMUNICATION DELAYS	217
APPENDIX C	THE ARPANET ROUTING ALGORITHM	221
APPENDIX D	SUM OF DELAYS FOR A COMPLETELY FILLED PRIMARY SUBTREE	224
APPENDIX E	A DISTRIBUTED FILE MIGRATION ALGORITHM	228
REFERENCES	242

LIST OF TABLES

<u>Table</u>		<u>Page</u>
3.1	History of Packet Arrivals for Hot Potato Forwarding . . .	90
3.2	Broadcast Forwarding Table for Network shown in Figure 3.2	92
4.1	Actual Performance of the Routing Algorithms in the ARPANET	160
4.2	Performance of Routing Algorithms in a Regular Graph of Degree 4 having 59 Nodes	161

LIST OF FIGURES

<u>Figure</u>	<u>Page</u>
2.1 Broadcast Along the Branches of the MST Initiated from Node 1	11
2.2a A Network	23
2.2b The Minimal Spanning Tree for the Network	23
2.3 A Part of a Network with Nondistinct Edge Costs	40
2.4a A Network	53
2.4b The Old Spanning Tree	53
2.4c New Leaves Being Created	53
2.5 All Nodes Become Leaves	61
2.6a An Old Spanning Tree with New Edge Costs	64
2.6b The New MST and a Possible Cycle FGON if a Computation Phase is Started Before the Current one has Ended	64
2.7 Undesirable Race Condition if Nodes Blindly agree on New Edge Costs	66
3.1a Shortest Path Tree from Node 5	83
3.1b Shortest Path Tree from Node 3	83
3.2 Multi-destination Addressing Along Shortest Paths from Node 5	85
3.3 Packet Propagation using Hot Potato Forwarding after 3 Hops	90
3.4 Suboptimal Reverse path Forwarding Initiated from Node 8	96
3.5 The Tables at Node 6 for the Network of Figure 3.2 to Achieve Optimal Reverse Path Forwarding	98
3.6 Broadcast Along the MST Initiated from Node 6	104

List of Figures

4.1	Some Regular Graphs	119
4.2	A Moore Graph	121
4.3	A Generalized Moore Graph Having 16 Nodes and Degree 3	122
4.4	A Pseudo Generalized Moore Graph Having 22 Nodes and of Degree 3	122
4.5	Lower Bound on the Diameter of Regular Graphs	126
4.6	Lower Bound on the Sum of Shortest Path Lengths in Regular Graphs	128
4.7	Lower Bound on the Average Path Length in Regular Graphs	128
4.8a	Shortest Path Tree for Routing	131
4.8b	Order in which Packets are Transmitted	131
4.9	Lower Bound on Maximum Broadcast Delay for Separately Addressed Packets	134
4.10	Lower Bound on Broadcast Cost for Separately Addressed Packets	134
4.11	Lower Bound on Average Broadcast Delay for Separately Addressed Packets	138
4.12	Lower Bound on Broadcast Cost for Multi-Destination Addressing and Source Based Forwarding	140
4.13	Upper Bound on the Number of Packets Transmitted for Hot Potato Forwarding with Discard Threshold	144
4.14	Lower Bound on the Number of Packets Transmitted for Hot Potato Forwarding with Discard Threshold	144
4.15	Number of Packets Transmitted for Simple Reverse Path Forwarding	147
4.16	Arrangement of Nodes at the Highest Level	151
4.17	Lower Bound on Broadcast Cost for Minimal Spanning Tree Forwarding	153
4.18	Lower Bound on the Number of Packets Transmitted for Regular Graphs of Degree 3 using the Broadcast Routing Algorithms	155

List of Figures

4.19	Lower Bound on Average Broadcast Delay for Regular Graphs of Degree 3 using the Broadcast Routing Algorithms	155
4.20	Lower Bound on Maximum Broadcast Delay for Regular Graphs of Degree 3 using the Broadcast Routing Algorithms	156
4.21	Lower Bound on Broadcast cost for Regular Graphs of Degree 3 using the Broadcast Routing Algorithms	156
4.22	The ARPANET Geographic Map, August 1976	158
4.23	The ARPANET Logical Map, August 1976	159
5.1	A Distributed Network Environment	168
5.2	A Distributed File System	168
5.3a	A Hierarchical Catalog Structure without Links	171
5.3b	A Hierarchical Catalog Structure with Links	171
5.4a	A Logical Catalog Structure	174
5.4b	Partitioning Based on Pointers in a DFS With 3 Hosts	174
5.5a	Logical Structure of the File System in the DCS	181
5.5b	Dotted Lines Superimpose a Possible Physical Organization	181
5.6	A Graph $G = [V, A(t)]$	188
5.7	A Loop is Produced if Axiom 5.2 is Violated	188
5.8	A Spurious RUD Entry at v_5 at Time $t+1$ Produces Loop in the Cascade-Search	188
B.1a	A Network	218
B.1b	The Minimal Spanning Tree for the Network	218
C.1	The ARPANET Routing Tables at an IMP	222
E.1a	Physical Neighbor Hosts of Host 1	231

List of Figures

E.1b	The Utilization Matrix at Host 1	231
E.2	Sub-optimal Decision using Algorithm I	234
E.3	Host 1 and its Physical Neighbors	235
E.4	Protocol Layering in the ARPANET	241

CHAPTER 1

INTRODUCTION

1.1 Introduction

This thesis investigates the design and analysis of broadcast protocols in packet switched computer networks. In particular, we are concerned with the design and analysis of broadcast routing algorithms for use in store-and-forward packet switching computer networks.

Computer communication network feasibility and utility has been demonstrated by the ARPANET [Roberts72, McQuillan72], and the other operational and planned networks like Telenet [Opderbeck76], TYMNET [Tymes71], CYCLADES [Pouzin73], and AUTODIN II. Much of the research accompanying these developments has focussed on the communication subnet itself [Fultz72, Gerla73, Metcalfe73, McQuillan74, Lam74, Tobagi74, Kamoun76]. The emergence of packet communication as an important technology for computer communication has been clearly demonstrated. Packet switching provides communication using computers as well as communication among computers.

We are only now beginning to see the sharing of resources between the computers connected by the communication subnet; one of the original goals of the ARPANET [Roberts70]. All communication between the host computers is viewed as interprocess communication, and resource sharing is made possible by a layer of protocols or agreements [Crocker72]. Interprocess communication is the basis on which all other protocols are

built and has received a great deal of renewed interest [Carr70, McKenzie72, Walden72, Metcalfe72, Cerf74, Sunshine75]. We believe that it is now time to examine broadcast protocols.

Distributed operating systems that support a distributed computing environment [Farber72a, Thomas73, Crocker75] may often have to make available to a user process a remotely resident resource. The resource may be capable of migration (e.g. files in a distributed file system or processes capable of performing specialized functions), or could be the least expensive copy of a duplicated resource [Cosell75]. In order to find such a resource the requesting host may have to send a request message to all hosts potentially capable of supplying the resource. In general, this set of hosts will be a subset of all the hosts in the network. For the purpose of this thesis, however, we consider the problem of delivering the message to all hosts. The requestor will be said to broadcast the message (to all hosts).

Broadcast protocols are, therefore, similar to interprocess communication protocols except that communication is between one process and many processes. Reliable broadcast protocol design is faced with problems similar to those in the design of interprocess communication protocols - addressing, sequencing, duplicate detection and guarantee of delivery.

The efficiency of the broadcast is greatly dependent on the nature of the particular subnet over which it is attempted. The structure of the subnet also influences the design of the broadcast protocol chosen to find resources. For example, multiaccess channels, like those

available in the ALOHA system [Abramson70], the Ethernet [Metcalfe76], satellite networks [Abramson73], or ring networks [Farber72] lend themselves very well to broadcast protocols since the very nature of the subnet makes every transmission available to all hosts. Circuit switched networks provide point-to-point communication, and so broadcast is done either by having a separate circuit between the broadcaster and each receiver, or by creating a multidrop circuit, that behaves like a ring, between the broadcaster and the receivers. Store-and-forward packet switched networks (PSNs) have storage and a (small) holding time at every switching node. PSNs are more suitable for performing broadcast than CSNs, as advantage can be taken of the packet mode of communication, and so separately addressed packets to each destination need not be transmitted. The ARPANET will be used as the model for PSNs. Some observations on broadcast protocols and routing in packet switched networks can be found in [Kahn76].

Multi-destination routing, which can be used for broadcast routing is found in AUTODIN I [Paoletti73], in the form of multiple addresses in the message headers. AUTODIN I is a message switched network and does not have the same stringent real time constraints as packet switched networks. The average address multiplicity per message in this network is 1.75. It is quite likely that multi-destination addressing is found in some other military, or special purpose networks (SITA), but this capability has not been well documented.

There are many ways of performing broadcast in PSNs so as to reduce the total amount of communication needed, thereby performing the

broadcast quickly and cheaply, as well as lowering the possibility of subnet congestion. We describe five alternatives to transmitting separately addressed packets to each destination. One of these ways is to use a multi-destination addressing scheme in the communication subnet. We determine the reliability of such algorithms to deliver packets to all hosts, and examine the effect of this on broadcast protocols at the host level. We propose a set of performance measures for broadcast communication and evaluate the algorithms by determining lower bounds for these measures in regular graphs.

We now briefly describe a few applications where broadcast routing is useful. First some terminology for describing the components of the distributed network environment. A host is a computer system that is a potential user and/or supplier of resources in the distributed computing system. A user is a person or a program that interacts with the host. A user process is a process associated with a user. A switching node is a device, in many cases a small computer, that accepts data and control from the host and sends it over the communication links, with the possible cooperation of other switching nodes, to the destination. The collection of switching nodes and communication links is the communication network or communication subnet. In a PSN, the switching nodes will be packet switches. We assume that all communication between hosts is viewed as interprocess communication.

In distributed data gathering systems, real time data may be generated by remote sensors. The data must be collected before it is processed. For reasons concerning reliability and simplicity the

sensors may not know which nodes, or how many of them are gathering data. Only the gathering nodes need decide which information to keep. Broadcast routing is necessary for such a scheme to work efficiently.

In the ARPANET user authentication and billing scheme [Cosell75], a TIP must locate an RSEXEC server process to verify the users' password, and then periodically record the users' usage of the network. Broadcast communication helps in locating an RSEXEC server faster, and can help in redundantly storing accounting data.

In distributed file systems, file access requests can be broadcast, so that the user need not know the current location of the file. If broadcast communication is efficient, then the structure of the catalog for the file system can be simplified. We describe how the catalog structure and search algorithms benefit by the availability of broadcast communication.

To create a very secure communication environment, broadcast communication can be used to send all messages to all destinations efficiently. By choosing the appropriate broadcast routing algorithm, uniform traffic patterns can be created within the subnet. By encrypting the data, only the appropriate receivers will be able to decode the message. As a result, an observer can not tell from where the data originated and to where it is destined.

In addition, in this thesis, we describe two distributed algorithms for constructing minimal spanning trees in computer communication networks, in which there is no one source of control. The algorithms

are both asynchronous and concurrent in operation. Our investigation into the design of such algorithms was spawned by our efforts to design broadcast routing algorithms. The minimal spanning tree can be used for forwarding broadcast messages. Each node knows which of the edges incident to it are branches of the minimal spanning tree. Hence, a "broadcast" packet arriving on one such branch would be delivered to all hosts connected to the node, and forwarded along the remaining branches. By being able to construct such trees in parallel, it is possible to adaptively construct new minimal spanning trees as the conditions within the subnet change. Minimal spanning tree routing also appears to have application in the design of adaptive routing algorithms, since the branches of the minimal spanning tree could be used to transmit delay estimates to all nodes, rather than using the hop by hop refinement technique [McQuillan74]. The adaptive version of the distributed algorithm has applications in communication networks like the Packet Radio Network (PRNET) [Kahn75, Frank75] in which the packet radio repeaters must configure themselves into a virtual topology when randomly placed in an operating environment, for example when dropped out of an airplane. The algorithms can also be used in a multiprocessor computer system to construct minimal spanning trees for the many applications they are used for. We believe that these algorithms are the first of their kind. The design of such algorithms reveal the kinds of communication and synchronization problems that are typical in distributed computing environments.

1.2 Summary

In Chapter 2, we present two distributed algorithms for the construction of minimal spanning trees in computer communication networks. The algorithms are based on the sequential algorithm proposed by Prim [Prim57]. The algorithms can be executed concurrently and asynchronously by the different computers of the network. There is no one point of control. The important feature in their design is the synchronization and communication between the processors, in an environment where there may be arbitrary delays in communication. The first algorithm is static, in that it assumes certain initial conditions and constructs the minimal spanning tree. The second algorithm is adaptive and executes continuously. It dynamically converts the old (minimal) spanning tree into a new minimal spanning tree when edge costs change, nodes are removed from the network, or new nodes are added to the network. Such algorithms can be used in multiprocessor computer systems as well.

Chapters 3 and 4 focus on the design and analysis of broadcast routing algorithms in store-and-forward packet switched computer networks. Chapter 3 proposes five alternatives to separately addressing packets to each destination, and qualitatively compares them. The reliability achievable from these algorithms is determined. The properties of a high level broadcast protocol emerge from the properties of the underlying routing algorithm and the requirements of processes using the protocol. We discuss the tradeoffs between a broadcast-to-all strategy compared to a broadcast-to-subset strategy.

Chapter 4 proposes performance measures for broadcast communication in such networks. The measures determine the overhead imposed on the communication subnet by a broadcast, and the delays in achieving broadcast. The various broadcast routing algorithms are quantitatively compared by determining lower bounds on their performance in regular graphs. The lower bounds determine the performance of these algorithms in "ideal" networks, against which network designers may measure their networks.

In Chapter 5 we discuss the structure of a distributed file system. We review current work in this effort, and propose a model to structure the catalog of the file system, and search algorithms for locating the files. We show how the performance of the system improves if there is an underlying broadcast communication capability. By using such a capability, it is not necessary to have multiple copies of the catalog, which must be kept consistent.

Chapter 6 wraps up this thesis by stating the results and conclusions of our research. Open problems stemming from our work are stated, since they provide important areas for future research. A set of Appendices enhance the description in various chapters. There is one Appendix per chapter - the first one provides a detailed table of contents to this thesis.

CHAPTER 2

DISTRIBUTED ALGORITHMS FOR CONSTRUCTING MINIMAL SPANNING TREES

2.1 Introduction

In this chapter, we present two distributed algorithms for constructing minimal spanning trees in computer communication networks. The algorithms can be executed concurrently and asynchronously by the different computers of a network. There is no one source of control. The algorithms are also suitable for constructing minimal spanning trees using a multiprocessor computer system. The first algorithm is static, in that it assumes certain initial conditions and constructs the minimal spanning tree. The second algorithm is adaptive and executes continuously. It dynamically converts an old (minimal) spanning tree into a new minimal spanning tree when edge costs change, nodes are removed from the network, or new nodes are added to the network. Some applications in computer communication networks that require constructing minimal spanning trees are now described.

The minimal spanning tree can be used for forwarding broadcast messages in distributed operating systems built on top of a packet switched network. If a minimal spanning tree was embedded on the existing subnet topology, then any node on this minimal spanning tree could initiate a broadcast, and the packets would be forwarded along this tree to all destinations. Each node knows which of the edges incident to it are branches of the minimal spanning tree. Hence, a

"broadcast" packet arriving on one such branch would be delivered to all hosts connected to the node, and forwarded along the remaining branches. This technique assumes, of course, that the cost of communication along an edge of the network is same in both directions. This is not true, in general, for PSNs, since the traffic patterns determine the queuing delays in either direction of a link. It is, however, not a bad approximation. Figure 2.1 shows the communication subnet of a PSN with the embedded minimal spanning tree. If broadcast was initiated from a host connected to node 1, then a packet would be transmitted along each of the minimal spanning tree branches in the directions shown in the figure. Note that all the edges in the subnet do not have the same cost. We examine, in detail, the suitability of using the minimal spanning tree for broadcast routing in Chapters 3 and 4.

Broadcast routing is also used by the minimal spanning tree algorithms themselves. When the minimal spanning tree has been constructed, all nodes must be informed and the simplest way is to broadcast the message along the branches of the tree. In the construction process itself, a message may have to be sent from one node to another in the same subtree as the originator. We shall see that we can use the subtree for routing, by broadcasting the message rather than relying on another underlying routing mechanism.

Minimal spanning tree routing also appears to have application in the design of adaptive routing algorithms, since the branches of the minimal spanning tree could be used to broadcast delay estimates to all nodes, rather than using a hop by hop refinement technique [McQuillan74]. We have not explored this possibility.

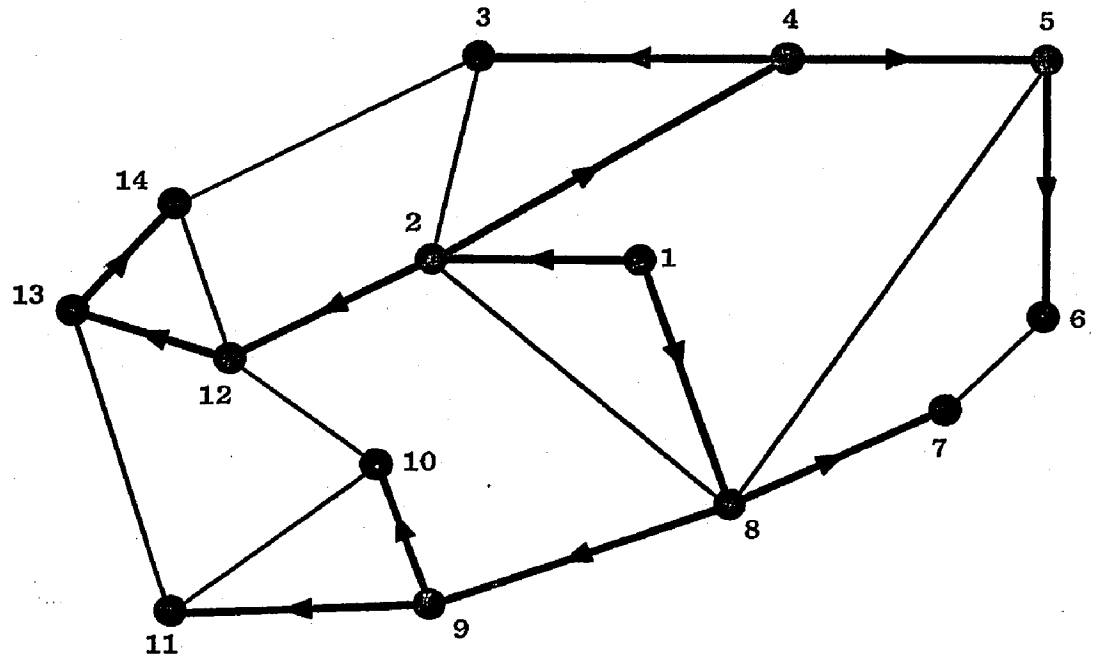


Figure 2.1. BROADCAST ALONG THE BRANCHES OF THE MST INITIATED FROM NODE 1.

The adaptive algorithm has special application in communication networks like the Packet Radio Network (PRNET) [Kahn75, Frank75]. The packet radio repeaters must configure themselves into a suitable topology when randomly placed in an operating environment, for example when dropped out of an airplane. Currently a tree-like topology is set up from a centralized point of control. If a minimal spanning tree is acceptable, then this algorithm could be used.

Section 2.2 discusses construction principles for minimal spanning trees, and section 2.3 describes an abstract parallel algorithm by which such trees can be constructed in a distributed environment. Sections 2.4 and 2.5 discuss the static algorithm, and 2.6 the adaptive algorithm. These algorithms are based on the abstract parallel algorithm.

2.2 Construction Principles for Minimal Spanning Trees -

In this section we review definitions and construction principles of minimal spanning trees. A network is composed of a set of nodes and a set of edges that connect pairs of nodes and have a cost associated with them. The Minimal Spanning Tree (MST) of such a network is a subset of the edges such that there exists a path between every pair of nodes, and the sum of the cost of these edges is a minimum. The edges in this MST will be called the branches of the MST.

In graph theoretical terms the MST problem can be stated as follows. Consider a connected, undirected graph, G , with vertex set V , and edge set, E (E is a subset of $V \times V$). A spanning tree is a subset of E , such that there is a unique path between any two vertices in V . Suppose there is a cost associated with every edge in E ; a minimal spanning tree of G is a spanning tree of G that minimizes the sum of the cost of the edges. [Bentley75].

The path between any two nodes in a spanning tree is the sequence of edges of the spanning tree that must be traversed to get from one node to the other. The cost of a path is the sum of the edges comprising the path. If the cost of a path is larger than that of another, then that path is said to be longer than the other. The diameter of a spanning tree is the cost of the longest path in the spanning tree.

Bentley and Friedman [Bentley75] briefly review existing techniques for the construction of MSTs and propose fast algorithms for the

construction of MSTs in multidimensional coordinate spaces. These algorithms are of the order of $N \log N$, where N is the number of nodes. A complete bibliography on the subject (up to 1974) can be found in [Pierce75]. The construction principles for MSTs were first formalized by Prim [Prim57] and are applicable to networks for which the edge costs (e.g. length, distance, delay) need not be distinct or consistent with Euclidean geometry.

The networks of interest to us will be the general class of networks studied by Prim.

2.2.1 Prim's Principles

Prim (1957) suggested two principles for constructing MSTs. We paraphrase some of his definitions, construction rules, and conditions. The principles assume that the construction process is sequential. An isolated node is a node to which, at a given stage of the construction, no connections have yet been made. A fragment is a subset of nodes connected by edges between members of the subset. These edges will become branches of the MST. An isolated fragment is a fragment to which, at a given stage of construction, no external connections have been made. The distance (cost) of a node from a fragment of which it is not an element is the minimum of its distances (costs) from each of the individual nodes comprising the fragment. A nearest neighbor of a node is a node whose distance from the specified node is at least as small as that of any other. A nearest neighbor of a fragment, analogously, is a node whose distance from the specified fragment is at least as small as that of any other.

Prim proved that an MST could always be constructed by following the following two principles.

Principle 1 (P1): Any isolated node can be connected to a nearest neighbor.

Principle 2 (P2): Any isolated fragment can be connected to a nearest neighbor by a shortest available edge*.

These principles were based on two necessary conditions.

Necessary Condition 1 (NC1): Every node in an MST must be connected to at least one nearest neighbor.

Necessary Condition 2 (NC2): Every fragment in an MST must be connected to at least one nearest neighbor by a shortest available edge.

2.2.2 Existing Algorithms

Most existing algorithms use P1 and P2 to create an isolated fragment and then increase the number of nodes in the fragment until it becomes an MST. The primary concern has been how to structure the data so that it is possible to quickly determine the shortest edge by which an isolated fragment can be connected to a node outside it. This is of

*The nearest neighbor of a fragment may be connected to the nodes of the fragment by more than one edge. Usually the process of determining the nearest neighbor of a fragment will involve examining the edge costs connecting nodes within the fragment to nodes outside it, and so the shortest available edge will easily be determined.

great importance if a fully connected network having a large number of nodes is under study. All these algorithms are sequential; there is no concurrency in growing many isolated fragments. The multi-fragment algorithm [Bentley75] is sequential in its operation.

The goal of our research is to describe concurrent, asynchronous algorithms to create an MST. Such algorithms are desirable not necessarily for the increased speed of execution (which we expect), but also because they ensure that there is no one source of control. Such algorithms are ideally suited to computer communication networks, in which control is distributed in order to improve the reliability of the network, and to make the network more sensitive to changing load conditions. The algorithms may also be used for constructing MSTs for other applications using a multiprocessor computer, such as the Pluribus [Ornstein75].

2.3 Parallel MST algorithms

In the most abstract setting, parallel algorithms, based on Prim's principles, for constructing MSTs can be stated as follows.

Initialization: Let every node be a fragment. A fragment is a subtree of the MST. The MST is unique if all the edge costs are distinct [Kruskal56].

General step: Choose one or more fragments. For each fragment chosen select the minimum cost edge connecting a node of the fragment to a node outside the fragment. Add all selected edges to the spanning tree being constructed. These added edges combine certain fragments. Combine these fragments into new, larger fragments.

Repeat the general step until only one fragment is left.

It is immediate from the basic properties of minimum spanning trees and from the fact that the MST is unique, that this algorithm correctly computes the MST. The difficulties in implementing the algorithm arise ~~from communication and synchronization problems.~~ We present in this chapter two techniques for performing communication and synchronization between the concurrent processes, and describe possible alternatives and their underlying assumptions. We call these algorithms distributed algorithms because the computers of a network are very loosely coupled, and because we would like the algorithms to be resilient to network errors and failures.

2.3.1 Properties of Distributed MST Algorithms

A distributed algorithm consists of a program executing in each of the nodes such that, when all the programs terminate, the result would be an MST connecting the nodes. Every node will know which of the edges connected to it are branches of the MST. It will be necessary for the nodes to communicate with their neighbors or some other node by means of messages. Properties of such algorithms that are of interest include:

- (i) Does shared information between nodes have to be locked when modified?
- (ii) What form of synchronization is required between the nodes?
- (iii) Are there any special initial conditions?
- (iv) Does the algorithm work only for certain combination edge costs?
- (v) In a network environment can the algorithm account for some of the nodes going down, new nodes coming up, edges breaking, or edge costs changing?

In order to prove that any algorithm for constructing MSTs works, it is sufficient to show that every operation performed is identical to P1 or P2, and that the algorithm terminates. We will constantly map the algorithms we describe, in terms of the abstract parallel algorithm in order to see how the synchronization and communication between the concurrent asynchronous processes is achieved. Note that for the

parallel algorithm based on Prim's principles to work, the edge costs in the network must be distinct so that the MST is unique. This in general will not be the case, and so we first describe a distributed transform by which a graph can be converted into one with distinct edge costs.

2.3.2 Transforming a Network into one with Distinct Edge Costs

Since there is only one edge connecting any two nodes in the network, and nodes have distinct identities (numbers), each edge has a unique pair of node identities associated with it. This makes it very easy to dynamically order the edges with the same cost, thus transforming the network into one with distinct edge costs. Spira suggested an idea that led to this transform, and the formal description, which follows, was suggested by Tarjan.

Let $C(e)$ be the cost associated with edge e , where e is a tuple (n_1, n_2) , where n_1 and n_2 are the identities of the two nodes. Let the modified cost $C'(e)$ of e be the triple

$$C'(e) = (C(e), \min(n_1, n_2), \max(n_1, n_2)).$$

The triples are ordered lexicographically, i.e. $(i, j, k) < (i', j', k')$ iff $i < j$, or $i = i'$ and $j < j'$, or $i = i'$ and $j = j'$ and $k < k'$. With modified costs defined in this way all edges have distinct costs, and $C(e_1) < C(e_2)$ implies $C'(e_1) < C'(e_2)$ where e_1 and e_2 are any two edges in the network. There is now a total ordering defined on the modified edge costs, and that is all that is necessary for the parallel MST algorithm to work.

This transformation can be performed every time a node compares the cost of two edges, since each node in a network typically knows the identity of the node at the other end of an edge. Hence, the network can be transformed into one with distinct edge costs using a distributed algorithm.

2.4 A Static Distributed MST Algorithm

This distributed algorithm can be used to construct an MST in a computer network, or using a multiprocessor computing system. The algorithm assumes that all components of the distributed environment are functioning correctly, and that each node knows which other nodes are its neighbors and the cost of the edges connecting it to its neighbors. This algorithm is not adaptive to changing edge costs or new nodes being added to the network, and hence the term "static".

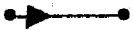
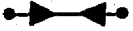

2.4.1 The Basic Model

The underlying philosophy of the algorithm is based on NCl, which states that every node must be connected to at least one of its nearest neighbors. Hence every node knows which neighbor to form a branch with. However, the result of such an action by every node will create an MST only in some cases. In general, such an action will produce a number of fragments that must be connected together appropriately. The distributed algorithm must discover that such a fragment has been created and then choose an appropriate edge to connect the fragment to other such fragments without introducing any cycles in the graph.

We now introduce some definitions, and prove some simple properties of the model. This is mainly to provide a framework and vocabulary for the treatment of the algorithm which will follow.

2.4.2 Definitions

Figure 2.2a shows a network, in which every edge has a distinct cost associated with it. The MST for this network is shown in figure 2.2b. Notice that some of the branches of this MST have been marked. The markings have the following interpretation:

- 
 Such a branch is called a singly marked branch. This branch is part of the MST since it connects the node from which the arrow emanates to its nearest neighbor (by virtue of NCI).
- 
 Such a branch is called a doubly marked branch. It connects both nodes to their nearest neighbors.
- 
 This branch is unmarked.

In figure 2.2b, edges BD, CF, GF, FJ, NM, EI, HK and PO are singly marked. Edges AD, IK, LO and JM are doubly marked while DE, EF and IL are unmarked.

The largest fragments composed only of marked branches (singly or doubly) will be called Marked Fragments (MFs). Notice that MFs are connected by unmarked branches to form larger fragments until the MST is formed. In figure 2.2b, unmarked branch DE connects marked fragments {A, B, D} and {H, K, I, E}. In a network that does not have distinct edge costs, a number of MSTs are possible. There may be different ways of creating MFs in each case, and so the number and identity of the singly and doubly marked branches will vary.

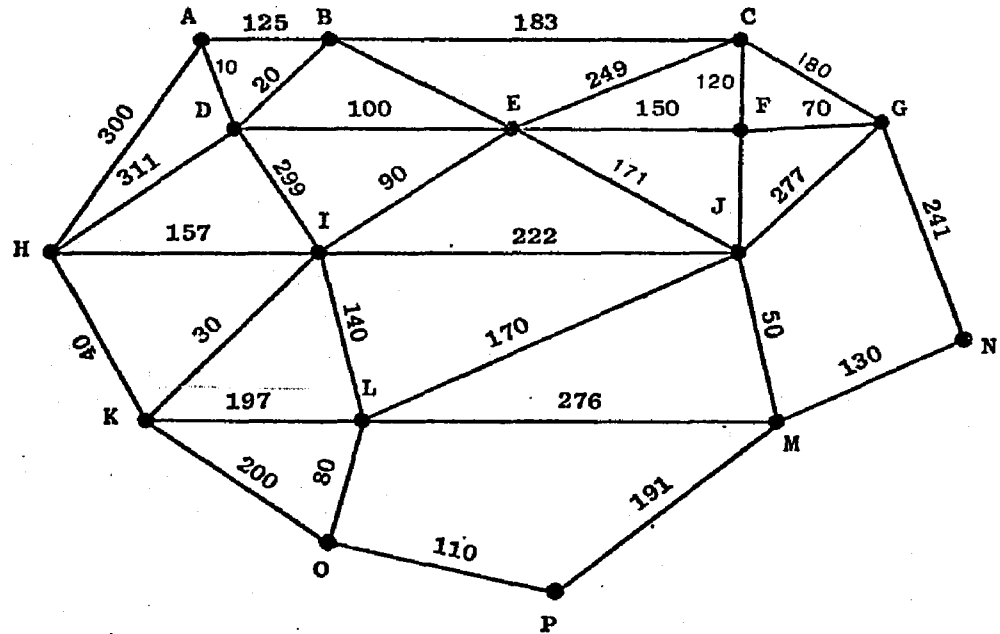


Figure 2.2a. A NETWORK.

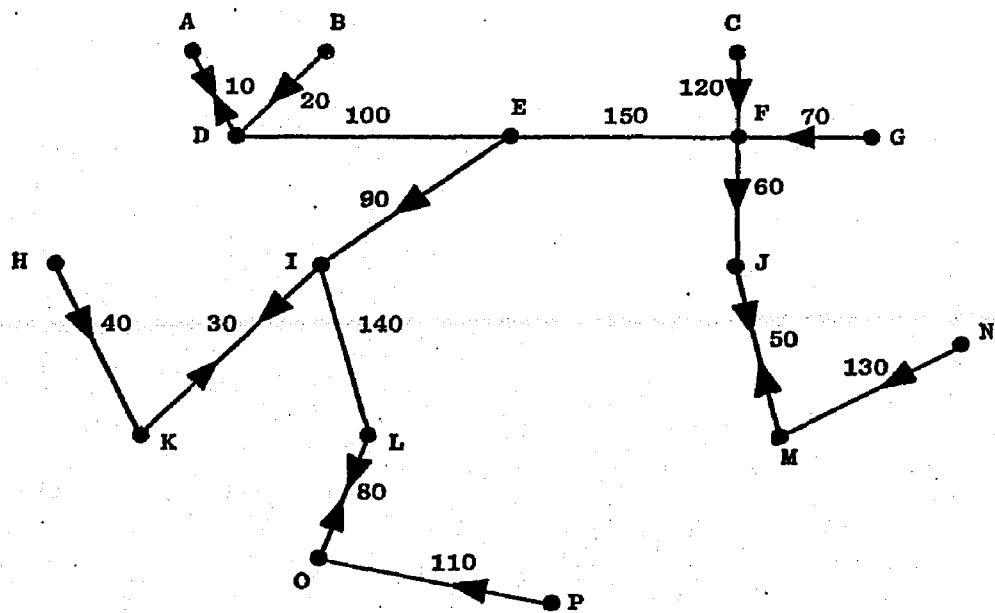


Figure 2.2b. THE MINIMAL SPANNING TREE FOR THE NETWORK.

We now state and prove some simple properties of these MSTs.

We know that a network with N nodes has an MST with $N-1$ branches.

Theorem 2.1: In an MST, the number of MFs equals the number of doubly marked branches, and each such fragment contains exactly one doubly marked branch.

Proof: Every MF is a subtree of the MST (by definition of the construction of MSTs). Each MF is also an MST for the nodes included in the MF (also by definition of MSTs).

Every node in the MF (let there be m of these) has a nearest neighbor. Thus, m branches will be marked. However, there are only $m-1$ branches in an MST of m nodes, so it follows that exactly one branch must be doubly marked. Thus, each MF has exactly one doubly marked branch.

The complete MST consists of a set of MFs, and the number of uniquely doubly marked branches is the cardinality of the set of MFs in the MST.

Q.E.D.

Corollary 2.1.1: Every MST has at least one doubly marked branch.

Theorem 2.2: In an MST, the number of unmarked branches is equal to one less than the number of MFs.

Proof: The marked fragments of the MST can be viewed as super nodes that are connected by unmarked branches to form a tree. A tree

with N nodes has $N-1$ branches, and so the number of unmarked branches in an MST is one less than the number of MFs.

Q.E.D.

A chain is a node subset (containing one or more nodes) connected by edges, that are singly marked. These edges connect nodes to their nearest neighbor. Edges are unique to a node, i.e. an edge cannot connect two nodes to each others' nearest neighbors. Such a chain is a fragment (but not a MF), and an MST for the node subset. The chain will be said to have one active node - the node that would connect itself to its nearest neighbor if the chain were extended, and still keep the fragment a chain. In figure 2.2b some of the chains and their active nodes are {GF; F active}, {G; G active}, {GF, FJ; J active}, {CF, GF, FJ; J active} and {CF, GF; F active}.

Notice that there is a certain monotonicity among the costs of branches in a chain. For every node in the chain, the cost of the branches incident at the node (as determined by the markings on the branch) is greater than or equal to the cost of the branch leaving the node (there is only one). This fact will be proved in the following theorem.

Theorem 2.3: The cost of the potential branch from the active node of the chain must be less than or equal to the cost of the branches in the starting chain.

Proof: This theorem is proved by induction.

If the chain consists of only one node (which is also active) then the cost of the potential branch must be less than those already in the chain (the null set).

Now assume that the chain has m branches satisfying this property. The active node has at least one branch incident at it. ~~The active node has an edge to its nearest neighbor which is not in the chain.~~ The cost of this edge can be less than or equal to that of the lowest cost incident branch, but not more otherwise the node at the other end of the potential branch would not be the active node's nearest neighbor.

Q.E.D.

Corollary 2.3.1: If the edge costs are distinct, then the branch out of the active node of a chain has a cost less than that of any branch in the starting chain.

If the active nodes of two chains decide they are each others neighbors, then the two chains merge, and this branch becomes a doubly marked branch of the resulting MF that these two chains are part of. The resulting fragment is no longer a chain.

When MFs connect to each other by an unmarked branch, the resulting fragment will be called a Minimal Spanning Subtree (MSS). A MSS becomes an MST when it contains all the MFs.

We now prove some simple properties for unmarked and doubly marked branches. The proofs will be made for networks with distinct edge

costs. The theorems will be valid for networks with this restriction removed except that the strict inequality will be replaced by a weaker inequality.

Theorem 2.4: For a network with distinct edge costs, the doubly marked branch of a MF has the lowest cost among the branches of that fragment.

Proof: The two nodes on either end of the doubly marked branch are active nodes of two chains which have all the nodes of the MF contained within them. The potential branch from these active nodes has a cost less than that for any branch in the respective chains (from Corollary 2.3.1). The cost of potential branches is the same since they are the same branch - a doubly marked branch. Hence the cost of a doubly marked branch is less than that of any other branch in the MF.

Q.E.D.

Theorem 2.5: For a network with distinct edge costs, the cost of an unmarked branch connecting two MFs is larger than that for the doubly marked branch of either MF.

Proof: The unmarked branch is connected to a node in the MF. This node is also connected to a marked branch in the MF and so the cost of the unmarked branch is greater (since edge costs are distinct) than that of the marked branch. From Theorem 2.4 it follows that the cost of this marked branch is greater than or equal to that of the doubly marked branch of the MF. Hence the cost of the unmarked branch is larger than that of the doubly marked branch. Since this

applies to both MFs connected by the unmarked branch, the theorem is proved.

Q.E.D.

These definitions and proofs are useful in understanding how the distributed algorithm for constructing an MST works, since the algorithm revolves around the ideas of concurrently creating MFs and having them grow into MSSs until the MST results.

2.4.3 Statement of the Algorithm

We now describe the algorithm for constructing an MST in a network with distinct edge costs. The MST in such a network is unique. In section 2.4.5 we show why the restriction on edge costs is necessary. It is, however, a simple matter to transform a network into one with distinct edge costs using a distributed technique (cf. section 2.3.2).

The basic philosophy of the algorithm is that each node must independently find its nearest neighbor and make the edge connecting it to that neighbor into a branch of the MST. The node then sends off a message to the neighbor informing it of this construction. Two nodes may realize that they are connected by a doubly marked branch. This is when the core of a MF is formed. This must grow into the MF. Such MFs will connect to other MFs or MSSs until an MST is created. Since the edge costs are distinct, the MST is unique. We will show in detail how MFs connect to other MFs or MSSs. No cycles are introduced by the asynchrony and concurrency of the computation, because a unique branch

connects any two fragments and there is never any ambiguity in choosing it.

We now introduce some more terminology. A node in a fragment is said to be the master if it decides from which node of the fragment, a branch should be created to a node lying outside the fragment. The node that actually makes the construction will become active. In a MF there is only one node that can be master. Initially there are no masters. When a doubly marked branch is created, one of the two nodes at either end unambiguously becomes master. We show later how this decision can be made. When two MFs are connected by an unmarked branch, there may be two potential masters (one in each MF). One of the masters unambiguously relinquishes control to the other, which then determines which node (of the fragment it has knowledge about) becomes active. The result of all this is an MST!

Every operation described so far has been consistent with Prim's Principles. We will show this more precisely a little later, and will now proceed to describe the algorithm formally.

2.4.3.1 State Information at Each Node

The statement of the algorithm will assume that each node has a set of state variables. These consist of:

- (1) The state of the node; i.e. whether it is inactive, active or master. The state determines what the node should do when it gets a message from another node.

(ii) Information about each of the edges the node is connected to. The information contains source and destination node identities of the edge, its cost, whether the edge is a branch, whether this node and/or the node at the other end marked the edge as a branch, and finally whether this node and/or the node at the other end made it into a branch without marking it.

(iii) A list of nodes that are in the fragment as seen by this node. For each such node, information is kept on edges (potential branches) connecting the node to nodes outside the fragment. Note that some of these edges could already be branches, since the node at which this data structure resides may not know to what other nodes the node at the other end of the branch is connected, and so can not include that node in the fragment state.

2.4.3.2 Internode Communication

Internode communication is achieved by sending messages called SIGNALS. Signals have five parameters; the source and destination node identities, the command, the fragment state as seen by the source of the signal, and, if an edge is being made into a branch, information about the edge as seen by the source of the signal. The command could be 'mark this edge', or 'become master' or 'become master and make this edge an unmarked branch'. A signal can be sent to a node that is a neighbor, or to a node that is not a neighbor but part of the same fragment, if the command is 'become master'. We assume that the communication link between two nodes is full duplex, and that an

underlying asynchronous mechanism guarantees reliable communication between two nodes.

2.4.3.3 Associated Routines

There are some special routines at each node. `MERGE_FRAG_STATE` merges the fragment state received in a signal with the fragment state already present at the node. Merging consists in adding nodes not already part of the fragment and deleting edges whose nodes now lie within the fragment. Note that the fragment state gets altered only when a signal arrives and is processed, and not when a signal is produced. Hence, when a node makes an edge into a branch, the fragment state is unaltered as this node does not know what lies beyond the node at the other end of the edge.

A routine called `MERGE_EDGE_INFO` merges the edge information received in the signal (if any) with that contained for this edge at the node.

`DECIDE` is a routine that determines which of two nodes should become master. If `DECIDE` returns true this node should become master. Relative node numbering could be used as an unambiguous decision. More esoteric techniques could be used which may help the algorithm execute faster. For example, both nodes know which edges the other is part of (since both nodes just exchanged fragment states). The node that becomes master could be the one that has a lower cost edge excluding the one that connects both together.

ANY_NEIGHBOR is a routine which examines the fragment state and determines which node (if any) should become master. If ANY_NEIGHBOR returns true, then the identity of this node is returned in MASTER_NODE, and the identity of the node at the other end of the edge from MASTER_NODE in DEST_NODE. The edge determined by (MASTER_NODE, DEST_NODE) connects this fragment to its nearest neighbor. If ANY_NEIGHBOR returns false, then there are no more neighbors of this fragment and thus the MST has been constructed.

TRANSFER_MASTER_CONTROL is a routine that examines the fragment state through ANY_NEIGHBOR. If there is a neighbor, and if MASTER_NODE is the node itself, then the node converts the edge determined by (MASTER_NODE, DEST_NODE) into a branch if it already was not one, and signals the DEST_NODE to 'become master and make this edge an unmarked branch'. If the edge is already a branch then DEST_NODE is signalled to 'become master'. If MASTER_NODE was not the node itself, then MASTER_NODE is signalled to 'become master'.

2.4.3.4 The Main Program

Abstractly, the program in each node can be formulated as follows.

Initialization: Determine the cost of the edges incident at this node and the identities of the nodes at the other end of the edges. Build the data structure corresponding to the edge information and the fragment state. The latter will contain only this node and its edges.

First step: Convert the edge to the nearest neighbor into a marked branch, and signal the neighbor to 'mark this edge'.

General step: Wait for a signal. When it arrives MERGE_FRAG_STATE and MERGE_EDGE_INFO.

If the command is 'mark this edge', then if the edge was marked by both nodes then DECIDE who should become master, as this is a doubly marked edge. If this node becomes master then TRANSFER_MASTER_CONTROL. If the edge was not marked by both nodes do nothing.

If the command is 'become master' then TRANSFER_MASTER_CONTROL.

If the command is 'become master and make this edge an unmarked branch', then if the edge was made into an unmarked branch by both nodes then DECIDE who should become master. If this node becomes master then TRANSFER_MASTER_CONTROL. If both nodes did not make the edge into an unmarked branch then TRANSFER_MASTER_CONTROL.

Repeat the general step.

2.4.4 Analysis of the Algorithm

We put off discussing the factors influencing the complexity of the algorithm till section 2.4.4.2, and now prove that it does in fact construct the MST.

In terms of the general model for parallel MST algorithms (cf. section 2.3 General step), when this algorithm starts, each node is a fragment and converts the edge connecting it to its nearest neighbor into a branch. The neighbor is informed of this by a message. This

message may incur a delay before arriving at its destination, and in the meantime the generator of the message is free to continue processing. Every node now waits for messages.

The arrival of messages determines which fragments have merged into larger fragments, and which nodes are permitted to repeat the general step based on their knowledge of the fragment.

If a message arrives announcing the establishment of a singly marked branch, then the node checks to see if it too had marked this branch. If not, then the node updates its data structures (merging fragments) and continues to wait for other messages. If this branch turns out to be a doubly marked branch, then the core of a MF has been created, and one of the two nodes unambiguously becomes master. There may be many such cores in creation in the network. This event is of great importance in the algorithm, since it determines which nodes can repeat the general step (cf. section 2.3). Recall that Marked Fragments are connected together by unmarked branches to create the MST. The master node is, therefore, now in search of an "unmarked branch" that will connect this MF to another MF or MSS. The decision on which edge to convert to a branch is based on the node's current information of the fragment. Since the MST is unique, so is the branch that connects a fragment to its nearest neighbor, and so even with the asynchrony in the operation of the algorithm, the decision of the active node is always correct. Note that this is true even when the signals take different amounts of time to be successfully transmitted. This is elaborated below.

In quest of this "unmarked branch" the node may pick an edge such that the node at the other end is part of the same MF. This is possible since the message from that node announcing the creation of the singly marked branch may not have yet arrived. Such an action is not harmful and is in fact important. Master control will be transferred to the new node, which will now grow the MF with the help of more complete fragment information, and master control will propagate until the "unmarked branch" to another MF or MSS is found.

A node that is master may even decide that an edge that has already been made into a branch (but still exists in the fragment state) connects the fragment to its nearest neighbor. The node just transfers master control to that node since it may have a more accurate view of the fragment and can make a better decision. The node which transfers master control can not pick another edge to convert into a branch because the edge it picks is not the lowest cost edge, and its inclusion as a branch can easily create a cycle.

Note that a node that is active may convert an edge into the "unmarked branch" without knowing what its complete MF looks like. This is not harmful since MF branches are always marked, and messages notifying neighbor nodes of this construction will eventually arrive.

Two active nodes may decide to make an edge into an unmarked branch simultaneously, in which case one of them unambiguously relinquishes control to the other, and the master grows this MSS.

Note that when the algorithm starts, there may be many nodes that are master nodes, but eventually this will decrease to zero. Master nodes will eventually determine that there are no more nodes lying outside their fragment, and will thus conclude that the MST has been created.

2.4.4.1 Termination of the Algorithm

The program at each node is said to terminate when it receives no more messages. Of course, the node does not know if it is going to receive any more signals or not, and so if it transmits data along the branches of the minimal spanning tree before it has been completely constructed, the data may not get to all destinations. The proof of the fact that the algorithm terminates is based on the observation that a new signal is sent (in response to one received, that has a command indicating that the node should become master) if and only if the fragment state at this node indicates that it is still possible to grow the fragment. Nodes which are told to become master will eventually refine their fragment states such that there will be no nodes lying outside the fragment and so no more signals will be generated.

In some applications it may be desirable for each node to explicitly know that the MST has been constructed. This is very easily achieved. There will be one or more master nodes that will discover upon examining their fragment states that there are no nodes lying outside the fragment, and therefore conclude that the MST has been constructed. These nodes can broadcast a signal whose command is

'done'. Upon receiving such a signal a node can proceed to use the branches of the MST.

2.4.4.2 Factors Influencing Complexity

The factors that influence the complexity of this algorithm are the degree of parallelism, the asynchrony of internode communication, the number of signals transmitted, the length of signals, the overhead of using a broadcast routing scheme to deliver signals to nodes in the same fragment, and the data structures representing the fragment state and edge information. All these factors are not independent of each other.

The determination of the complexity of parallel and distributed algorithms is very difficult. Not only does the complexity as a function of the number of processors and number of nodes have to be determined, but also the effect of asynchrony and delay in internode communication. There are no formal analysis techniques to be used in this case. The simulation of the algorithm under various assumptions and performance measures seems to be the best way to get a measure of the complexity of such algorithms. While we have simulated the algorithm to make sure that it does in fact work, the determination of its complexity is out of the scope of this thesis and a subject for future research. We do, however, discuss the effects of delays in transmission of signals, and how the length of signals change as the computation progresses. In section 2.5.1 we discuss alternate ways for gathering information about the fragment.

2.4.5 Networks in which the Edge Costs are not Distinct

The algorithm presented in the previous section constructed an MST since the edge costs were distinct, and so the decision made by an active node to convert an edge into a branch was always correct and unaffected by the asynchrony of the computation.

When the edge costs are not distinct, the asynchrony of the operation may introduce cycles, and thus will not construct an MST. To see why this is possible, consider the example shown in figure 2.3. Nodes A, B, and C are part of a larger network. Edges (A,B), (B,C) and (A,C) are all of the same cost. It may so happen that when each node is converting an edge into a branch using P1 that node A chooses B, node B chooses C, and node C chooses A. A cycle has resulted.

Similarly, if there are two MFs that have more than one possible unmarked branch connecting them together, then the master node in each MF may choose a different edge to convert into a branch, thus creating a cycle. Generalizing, we can say that if there is more than one edge that can be converted into a branch so as to connect two fragments together, then there is the possibility of a cycle.

Prim (1975) showed that if there are many edges of the same cost connecting a fragment to its nearest neighbors, then it did not matter which was chosen, and an MST would still be constructed. Therefore, if the network is converted into one with distinct edge costs, either implicitly (cf. section 2.3.2), or explicitly, then the algorithm presented in section 2.4.3, would still be suitable.

Recall that a node updates its knowledge of the fragment it is in only when it receives signals. Therefore, the master node makes a decision based on its current knowledge of the fragment. The decision could have been made just before a signal that has new fragment information arrives. The decision of the master node is never wrong or harmful in constructing the MST, but it might result in redundant processing and transmission of further harmless but unnecessary signals. Appendix B illustrates an example of such redundant computation being performed. Such effects must be taken into account when determining complexity.

Recall again, that the fragment state transmitted in signals consists of all the nodes within the fragment as seen by the originator of the signal, and for each such node, edges (with their costs) that connect them to nodes outside the fragment. The size of this data structure depends on the topology of the network and the connectivity of the nodes within it. Typically, when the fragment is small there will be few nodes within the fragment, but many potential branches. As the fragment grows the number of nodes within the fragment increases as does the number of potential branches, until a point is reached when the number of potential branches starts decreasing again. An interesting subject for future research is to determine what this variation is, as a function of network topology and internode communication asynchrony.

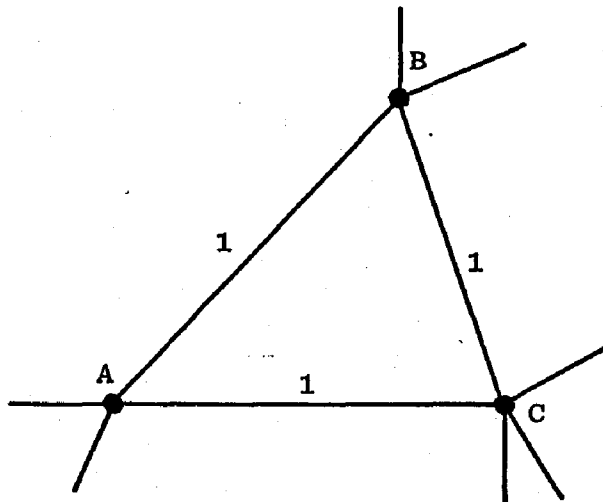


Figure 2.3. A PART OF A NETWORK WITH NONDISTINCT EDGE COSTS. The possibility of a cycle exists.

2.4.6 Conclusions

An algorithm has been described for constructing an MST in a computer communication network, or using a multiprocessor. This algorithm is asynchronous and concurrent, and so can be thought of as a parallel algorithm for constructing an MST.

The algorithm has the following properties:

(i) Information on the state of an edge is duplicated at both nodes of the edge. The data structure reflecting this information need not be locked at the other end when either node decides to modify it.

(ii) Cooperation between the nodes for the purpose of creating branches, and for refining the state of the fragment at each node is achieved by sending messages from a node to either its neighbor or to another node in the same fragment. There is no other need for synchronization between the cooperating processes, such as receiving positive acknowledgements for the messages, or aborting a previously "transmitted" message before it actually gets transmitted (cf. section 2.5). Transmission of messages to a node that is not a neighbor but in the same fragment, can be done by broadcasting (or relaying) it along the branches of the MST of this fragment. Hence there is no need for another routing scheme. We assume the existence of an underlying reliable interprocess communication mechanism for transmitting messages between nodes [Cerf74, Cerf74a, Sunshine75].

(iii) The only special initial condition is that all nodes know the cost of the edges connecting them to other nodes, and the identities of these nodes.

(iv) The algorithm is able to construct an MST in a network that has no constraint on the edge costs, since a network can always be transformed into one with distinct edge costs.

(v) The algorithm cannot incrementally account for nodes going down, edges breaking or changing costs, or nodes coming up. The MST has to be recomputed.

2.5 An Alternate Model

Tarjan (in a private communication to the author) suggested an alternate model, similar to the one presented in section 2.4, by which nodes communicate with one another and construct the MST. We discuss Tarjan's proposal and examine the relevance of the assumptions.

The initial conditions for this model are the same as those for the model of section 2.4, i.e. each node knows the identity of its neighbors and the cost of the edges connecting it to them. The edge costs are also distinct. In addition, assume that the communication across an edge takes place in one direction at a time. Thus, if both nodes at the end of an edge want to talk to each other simultaneously, one gets precedence (it doesn't matter which). We assume that there is a mechanism by which two nodes detect that both are attempting to use the communication channel simultaneously, and one wins. We examine the consequence of this assumption on the structure of the processes residing in each node, and the asynchrony of the entire system a little later.

Each fragment is defined by a set of edges previously added to the MST which form a subtree spanning the fragment. We shall direct these edges (branches) so that they form a rooted tree -- each node will have a single exiting edge, except for a unique node, the root, which has no exiting edge. The root will be the communications center for the fragment.

Each fragment computes independently. Only when the fragment tries to connect itself to another fragment via a minimum cost edge, or when another fragment tries to connect itself to the given one, is there interaction between fragments.

The computation carried out by a fragment consists of two steps.

1. Information Gathering: The root (i.e. the fragment's communication center) must find out the minimum cost edge connecting the fragment to another fragment. The root could broadcast a signal to all nodes in the fragment. A signal is then returned from each node in the fragment to the root. This information is combined, either along the way or in the root, to determine the minimum cost edge (say (X,Y)) connecting a node (say X) in the fragment with a node (say Y) outside the fragment.

2. Adding an Edge: The root sends a signal to X to add (X,Y) to the MST. This signal travels along a path of the subtree from the root to X . All edges along this path have their directions reversed. If (X,Y) has not yet been added to the MST, it is added with direction from X to Y . This fragment is thus combined with another fragment, and the new root is in some other part of the new fragment. If (X,Y) has already been added to the MST (direction from Y to X), X becomes the new root of the enlarged fragment, and step 1 is initiated from X . Hence, whenever a node gets a signal to create a branch, it does so if it is not already there, and determines if it is the root.

The process continues within each fragment, until one fragment has no potential branch connecting it to other nodes. The fragment is then the only fragment (if the graph is connected) and it contains all the nodes. The algorithm is now said to terminate.

There are several ways of doing the information gathering and edge addition, depending on how much information we wish to send along edges. These are examined in detail in section 2.5.1

The advantage of not permitting communication to occur simultaneously over an edge is that two nodes never convert the same edge into a branch simultaneously. Thus, the problem of deciding which node should become the root (analogous to deciding which of two nodes should unambiguously become master (cf. section 2.4.3)) never arises. Further, the asynchrony of the system is restricted such that no redundant computation is performed, as might occur if a master node created an unmarked branch out of an edge that was to be singly marked, and master control was "unnecessarily" transferred to the node at the other end.

The advantage of directing the branches is that when fragments connect to one another there is one resulting root. The algorithm thus proceeds in an orderly fashion, and there is no redundant computation. The way the MST evolves depends on the asynchrony of the communication between the nodes. In the algorithm described in section 2.4, master control was transferred from node to node, with the number of masters decreasing as fragments connected to one another. However, as we

noticed, if there are unpredictable delays then a lot of redundant computation may be performed as master control gets passed around unnecessarily (see Appendix B). The static distributed MST algorithm keeps no history of how master control is transferred and so redundant computation is possible.

Let us now examine the consequence of the half duplex nature of internode communication, and the fact that branches must be directed, on the implementation of such a scheme in a computer network.

In order that simultaneous internode communication not take place, it is not sufficient to treat the communication links as half duplex. This implies that if one node has already initiated an asynchronous transmission of a signal over a branch, and the branch is in use in the opposite direction, then this signal must be aborted, because the signal that arrives must be processed before anything else is done. A previous decision made by the the process must be undone. Alternatively, the process must acquire "ownership" of the channel in a manner similar to getting permission to enter a critical section of code. This may be possible in some implementations, but if the communication link is being multiplexed such "ownership" may be difficult to implement, if at all. In packet switched networks like the ARPANET there is a positive-acknowledgement retransmission protocol to guarantee reliable transmission between the switching nodes; the IMPs or TIPs. Ownership of the channel would have to extend until the signal was successfully transmitted. We notice, therefore, that the assumption of half duplex internode communication affects the asynchrony of the software in each node, and thus acts as a "synchronizing" mechanism between two nodes.

When a signal is sent to a node within the fragment, in order to connect the fragment to another fragment using a specified edge, the branches along the path the signal traverses must have their directions reversed. Hence, an underlying routing mechanism must exist that knows what the subtree of the fragment looks like, so that it can route the signal over appropriate branches. An underlying routing algorithm, like that present in the ARPANET may be useful, but again may not if it does not route the signal over the right path. Each node along the way must be informed that it must reverse the direction of the branch. Hence, it is not only necessary for the signal to reach its destination but for each node along the path to examine this signal (or an equivalent scheme).

2.5.1 Ways of Gathering Information about the Fragment

We describe some ways the root gathers information about the fragment of which it is the communications center.

The first approach is very similar to the one used in the static distributed MST algorithm described in section 2.4.3. At each node n , we store a set $F(n)$ which contains the nodes that n knows to be in the fragment, and some representation of the edges incident to nodes which n knows to be in the fragment. The root always chooses the minimum cost edge, say (X,Y) connecting a node X , in the fragment, to node Y outside. The root signals X to create this branch. All the information about the fragment as seen by the root ($F(n)$ and incident edges to nodes outside the fragment) must be passed in the signal, and the direction of the

branches traversed changed. Note that initially all nodes are roots of the fragment consisting only of themselves.

The second approach is to store only the set $F(n)$ at every node n and, of course, information on edges and nodes incident to n . The signal from the root indicating that a branch should be created contains a boolean vector representing $F(n)$. When a node discovers it is the root, it broadcasts a message (signal) to all nodes in the set $F(n)$ it has knowledge of, in order to get back from each of them, the cost and identity of potential branches leaving the fragment $F(n)$. The root then signals X to create the branch. This technique causes more signals to be generated because all the nodes in a fragment must be interrogated before the minimum cost edge leaving it can be determined. The amount of information in each signal is less than the first case. The responses to a broadcast query message from the root could be combined in the root or along the nodes in the subtree on the way to the root.

The third and last approach is for each node only to have information about the edges and nodes incident to it. When a node discovers that it is the root, it must determine the nodes within its fragment and the potential branches to nodes outside the fragment. The root could broadcast a message to determine which nodes are within the fragment and then broadcast a message to them to find out what edges leave the fragment and their costs. Both these operations could be performed in one stage rather than two. The length of each signal is now even smaller, and when the root signals X to create a branch, it does not include any information about the fragment. Notice that when a

root attempts to determine the identity of the nodes within its fragment, it is not sufficient for each of the nodes to send a message directly back to the root, because the root does not know how many nodes there are going to be within the fragment and so does not know how long to wait. Information must be combined along the tree back to the root. This is necessary because only a node that knows whether it is the leaf of the subtree spanning the fragment.

The second and third alternatives have less information in each signal but cause the generation of more signals than the first alternative. However, there is more processing overhead at each of the nodes in order to determine what the fragment looks like.

The second and third alternatives could have been used even with the static distributed MST algorithm described in section 2.4.3, since these alternatives do not depend on the assumption of half duplex internode communication, or the notion of directing the branches of the subtrees being created.

We leave the determination of the suitabilities of these various fragment information gathering schemes for future research, since their suitabilities are based on a measure of complexity of the resulting algorithm; a subject out of the scope of this thesis. We, however, feel that the assumptions of half duplex internode communication and the overheads for directing the branches may not be appropriate for certain applications in computer communication networks.

2.6 An Adaptive Distributed MST Algorithm

In this section we present an adaptive distributed algorithm for constructing MSTs. The algorithm executes continuously, and dynamically converts an old (minimal) spanning tree into a new MST. The algorithm, therefore has phases; each phase transforms a spanning tree into an MST. We show that one phase must be over before the next starts, otherwise the information each node has about the fragment it is in, may become inaccurate. If fragment information becomes inaccurate, then the construction process could easily produce cycles instead of constructing the MST. We will also describe how the algorithm moves from phase to phase.

The algorithm takes into account edge costs changing, new nodes being added to the network (edge costs to these nodes going from infinity to a finite value), and nodes going down (edge costs to them becoming infinite). The algorithm does, however, assume that all the nodes and edges are functioning correctly during a computation phase; i.e. nodes and edges can not go down during a computation phase, though new nodes and edges can be added. This algorithm can even be used to construct an MST from scratch because the nodes of the network can all be considered connected to a fake node. In this case, the algorithm looks very similar to the static distributed MST algorithm described in section 2.4. We examine this special case in section 2.6.4.

2.6.1 The Basic Model

Recall the abstract parallel MST algorithm described in section 2.3. Fragments (subtrees of the MST) connect to other fragments to create larger fragments until the one remaining fragment spans the entire set of nodes. Various implementations of this algorithm impose constraints on which fragments can connect to other fragments, the node within a fragment that makes the decision to connect to another fragment, and the form of internode communication and transfer of control. The algorithm described in section 2.4 transferred master control from node to node, while the one in section 2.5 determined which node should become the root when two fragments were connected to one another by the minimum cost edge, to form a larger fragment.

The algorithm to be described in this section uses the old spanning tree to systematically pass control around so as to transform this spanning tree into the MST. A tree spanning a network with N nodes, has $N-1$ branches. The algorithm requires that each node (save one) change one of these $N-1$ branches (one that is incident to it) into a branch of the new MST, thereby completing the transformation. The terminology used to describe the algorithm is similar to that used in section 2.4. We will repeat definitions when appropriate.

A node is said to be the leaf of the old spanning tree if it is connected to the old spanning tree by only one branch. In figure 2.4b A, C, F, H, K, M, N and P are the leaf nodes of an old spanning tree for the network in figure 2.4a. A node is said to be master if it decides

from which node of the new-fragment a branch should be created to a node lying outside the fragment. The node that actually makes the construction will become active. When a node becomes a leaf, it must remove the old-branch connecting it (and the fragment of the new MST) to the old spanning tree, and replace it with a new-branch that connects the new-fragment to its nearest neighbor. In this way the old spanning tree is converted into a new MST.

A node becomes master when it becomes a leaf node. The last branch of the old spanning tree is removed and a new one found. The master node may transfer master control to some other node in the fragment in search of the edge that connects this fragment to its nearest neighbor. This may recur till a node becomes active. A new-branch is created and master control disappears. However, if two active nodes convert the same edge into a branch, then one of them must unambiguously become master again. This is because a master node and therefore an active node was the result of removing an old-branch and replacing it with a new one. Each active node must create a branch, and so if two of them create the same branch, there is one less new-branch and one of the two active nodes must become master again.

Initially, the only fragments that can compute are those that consist of single nodes that are leaves of the old spanning tree. These nodes, asynchronously and concurrently, remove the branch that connects them to the old spanning tree and create a new-branch that connects them to their nearest neighbors. As a consequence of such an action by leaf nodes, other nodes in the network will become leaf nodes and the process

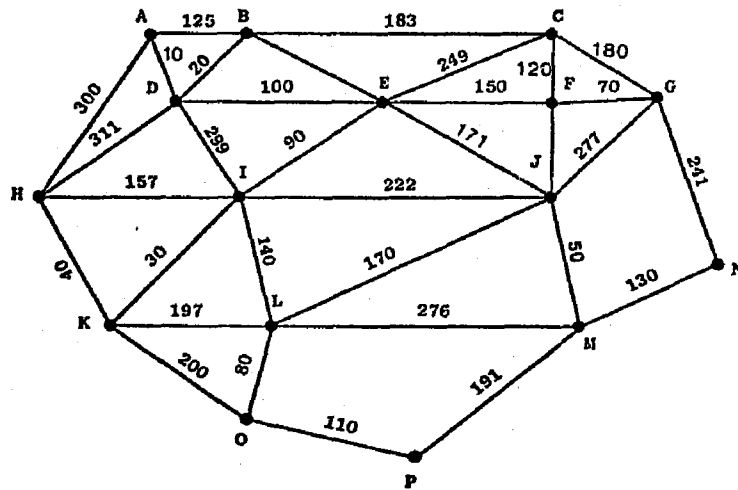


Figure 2.4a. A NETWORK.

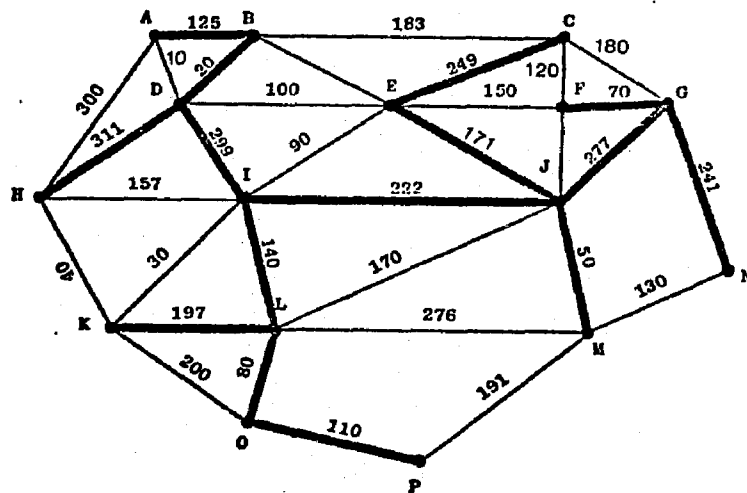


Figure 2.4b. THE OLD SPANNING TREE.

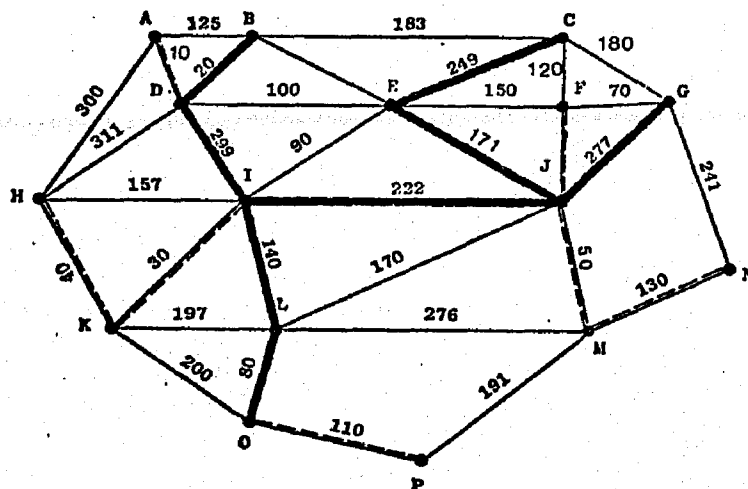


Figure 2.4c. NEW LEAVES BEING CREATED.

of replacing old branches continues, until all $N-1$ branches of the old spanning tree have been replaced by new ones. This results in an MST. In figure 2.4c, B, G and O are the newly created leaves after nodes A, F, K, H, M, N and P have replaced the old-branches incident to them. Signals that transfer master control or create a branch contain information about the fragment as seen by the originator of the signal. Note that if a leaf and therefore a master node wishes to make an old-branch into a new one, it need not remove the old one and then create the new one in two separate signals. The node could just create the new one, thereby implicitly removing the old one.

All N nodes in the network will eventually become master, but there are only $N-1$ branches to be replaced. Some node will be the last one to become master. It will upon examining its fragment state realize that there are no nodes lying outside the fragment, and therefore conclude that the fragment spans all the nodes. Therefore the MST has been constructed. Note that it is not necessary for each node to know the names of all the nodes in the network, or even the total number to make this decision. Hence, new nodes can easily be added to the network. We prove in section 2.6.3 that there is only one node that discovers this fact (unlike the algorithm of section 2.4.3). This node could then broadcast a 'done' signal to all other nodes along the branches of the MST. Upon receiving a 'done' signal a node knows that the last computation phase is over and that it can reinitialize its state information - for example, the new MST now becomes the old spanning tree, and edge costs must be reestablished. Once a node has

reinitialized itself, it can proceed with the next computation phase when it becomes a leaf without having to wait for all other nodes to have got the 'done' signal. We examine how a computation phase terminates and the next one initialized in detail in section 2.6.3.

2.6.2 Statement of the Algorithm

We now describe the algorithm formally. We assume that the only change between two phases is the edge costs changing. We assume that nodes and edges do not come up or go down. The algorithm accounts for these variations, but we leave it for section 2.6.5 to explicitly show this.

2.6.2.1 State Information at Each Node

The statement of the algorithm will assume that each node has a set of state variables. These consist of:

(i) The state of the node with respect to the construction of the new MST; i.e. whether it is inactive, active, master, or done, and the state of the node with respect to the old spanning tree; i.e. whether it is a leaf or not. The state determines what the node should do when it gets a message from another node.

(ii) Information about each of the edges the node is connected to. The information contains source and destination node identities of the edge, its cost, whether the edge is an old-branch or a new-branch, and if a new-branch whether this node and/or the node at the other end made it into a branch.

(iii) A list of nodes that are in the fragment as seen by this node. For each such node, information is kept on edges (potential branches) connecting the node to nodes outside the fragment. Note that some of these edges could already be branches, since the node at which this data structure resides may not know to what other nodes the node at the other end of the branch is connected to, and so can not include that node in the fragment state.

2.6.2.2 Internode Communication

Internode communication is achieved by sending messages called SIGNALS. Signals have five parameters; the source and destination node identities, the command, the fragment state as seen by the source of the signal, and if an edge is being made into a branch, then information about the edge as seen by the source of the signal. The command could be 'remove this old-branch', 'make this edge (old-branch) into a new-branch', or 'become master'. If the command is 'remove this old-branch', then the fragment state should not be present in the signal. A signal can be sent to a node that is a neighbor, or to a node that is not a neighbor but part of the same fragment if the command is 'become master'. We assume that the communication link between two nodes is full duplex, and that an underlying asynchronous mechanism guarantees reliable communication between two nodes.

2.6.2.3 Associated Routines

There are some special routines at each node. Many of them are identical or very similar to those defined in section 2.4.3.3. We choose to give these the same names because they perform the same basic functions.

`MERGE_FRAG_STATE` merges the fragment state received in a signal (if any) with the fragment state already present at the node. Merging consists in adding nodes not already part of the fragment and deleting edges whose nodes now lie within the fragment. Note that the fragment state gets altered only when a signal arrives and is processed, and not when a signal is produced. Hence, when a node makes an edge into a branch, the fragment state is unaltered as this node does not know what lies beyond the node at the other end of the edge.

A routine called `MERGE_EDGE_INFO` merges the edge information received in the signal (if any) with that contained for this edge at the node.

`DECIDE` is a routine that determines which of two nodes should become master. If `DECIDE` returns true this node should become master. Relative node numbering could be used as an unambiguous decision. More esoteric techniques could be used which may help the algorithm execute faster. For example, both nodes know which edges the other is part of (since both nodes just exchanged fragment states). The node that becomes master could be the one that has a lower cost edge excluding the one that connects both together.

ANY_NEIGHBOR is a routine which examines the fragment state and determines which node (if any) should become master. If ANY_NEIGHBOR returns true, then the identity of this node is returned in MASTER_NODE, and the identity of the node at the other end of the edge from MASTER_NODE in DEST_NODE. The edge determined by (MASTER_NODE, DEST_NODE) connects this fragment to its nearest neighbor. If ANY_NEIGHBOR returns false then, there are no neighbors of this fragment and thus the MST has been constructed.

CHANGE_BRANCH is a routine that gets invoked when a node becomes a leaf of the old spanning tree. The routine examines the fragment state through ANY_NEIGHBOR. There must be a nearest neighbor. If the edge (MASTER_NODE, DEST_NODE) is the same as the last old-branch, then the node makes this edge into a new-branch and signals DEST_NODE to 'make this edge (old-branch) into a new-branch'. Otherwise the old-branch is removed and DEST_NODE signalled to 'remove this old-branch', and a new branch is created by calling TRANSFER_MASTER_CONTROL.

TRANSFER_MASTER_CONTROL is a routine that examines the fragment state through ANY_NEIGHBOR. If there is a neighbor, and if MASTER_NODE is the node itself, then the node converts the edge determined by (MASTER_NODES, DEST_NODE) into a branch if it already was not one, and ~~signals the~~ DEST_NODE to 'make this edge (old-branch) into a new-branch'. If the edge is already a branch then DEST_NODE is signalled to 'become master'. If MASTER_NODE was not the node itself, then MASTER_NODE is signalled to 'become master'. If ANY_NEIGHBOR returns false, then broadcast a 'done' signal to all the nodes, and set the state of the node to done.

2.6.2.4 The Main Program

Abstractly, the program in each node can be formulated as follows. Assume that all nodes know who their neighbors are.

Reinitialization: Determine the new cost of the edges incident at this node. Reinitialize the data structure corresponding to the edge information and the fragment state. The latter will contain only this node and its edges. The edge information will reflect which edges are old-branches.

First step: If the node is a leaf then CHANGE_BRANCH.

General step: Wait for a signal. When it arrives MERGE_FRAG_STATE and MERGE_EDGE_INFO.

If the command is 'remove this old-branch', then do nothing more since the edge information already reflects this change.

If the command is 'become master' then TRANSFER_MASTER_CONTROL.

If the command is 'make this edge (old-branch) into a new-branch', then if the edge was made into a new-branch by both nodes then DECIDE who should become master. If this node becomes master then TRANSFER_MASTER_CONTROL.

If the command is 'done' then this computation phase is over, and so forward the 'done' signal along all the branches incident to this node, except the one on which the signal arrived.

If the node is now a leaf then CHANGE_BRANCH.

If this node is done then this computation phase is over.

Repeat the general step until this computation phase is over.

Repeat the Initializations.

2.6.3 Termination and Reinitialization of Computation Phases

We now prove, informally, that there is one and only one node in the network that determines that the MST has been constructed. This node informs the others of the termination of the current computation phase by broadcasting a 'done' signal, along the branches of the new MST just constructed. Recall that the algorithm constructs the MST by replacing the $N-1$ old-branches by new-branches. Old-branches get replaced when a node becomes a leaf. There are N nodes in the network and each gets the opportunity to be a leaf. We first show why all nodes eventually become leaves, as the proof that only one node determines termination of the computation phase depends on this fact.

As old-branches are replaced by new ones, other nodes become leaf nodes. Eventually, there will be one node, say Z (that has never been a leaf), that is connected by old-branches to m ($m \geq 2$) leaf nodes. See figure 2.5. These m nodes will attempt to replace the old-branches incident to them, and signals will be sent to Z . Nodes always process signals sequentially and to completion, and so when Z has processed $m-1$ signals it too will become a leaf. Therefore all nodes become leaves of the old spanning tree.

As a consequence of all nodes becoming leaves and there being only $N-1$ old-branches, after the last node has become a leaf, two master nodes in two fragments will make the same edge into a new-branch. (This

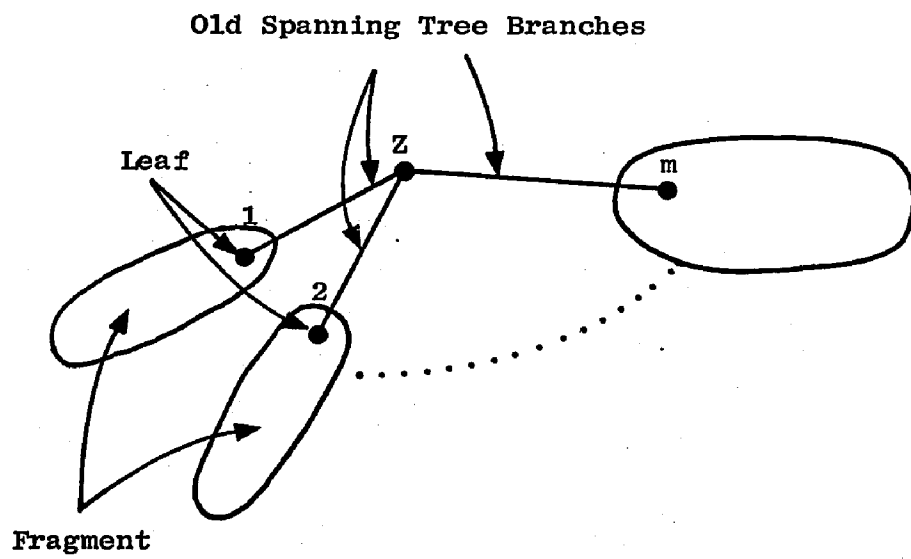


Figure 2.5. ALL NODES BECOME LEAVES.

is the last edge to be made into a new-branch by two active nodes.) One of these two nodes unambiguously becomes master again, and transfers master control in an attempt to connect the fragment it is in to another fragment. Master control will be transferred to some node that realizes that there are now no more nodes lying outside the fragment. This node, therefore, concludes that the MST has been constructed. There are no other master nodes in the network at this time, because all other master nodes, given rise from leaf nodes, have relinquished the "master" property once a new-branch was created.

Let us now examine how nodes reinitialize themselves for the next computation phase once a 'done' signal has been broadcast. A sufficient condition for this adaptive algorithm to work is that a new computation phase not be initiated while one is currently in progress. The reinitialization process must always guarantee that this condition is true. Let us see what happens if, for any reason, this condition is violated. Figure 2.6a shows a network with an old spanning tree and new edge costs. Assume that all nodes know that the last computation phase has terminated and have reestablished the new edge costs. Figure 2.6b illustrates what the new MST would look like. We now show that while this computation phase is in progress, if another one is initiated, then a cycle is constructed because different parts of the network have different and inconsistent information about the fragments being constructed. Consider that instant of time when B, C and D are leaves. C will remove branch CA and signal O to become master so that it will construct ON as a branch. Say nodes M and N decide to start another

computation phase and construct MF and NF as branches. F will become a leaf. At this instant assume that the cost of edge FB has become 29 from 10, and that of FG has become 16.5 from 29. F will remove FB and create FG as a branch. A cycle consisting of edges FG, GO, ON and NF has been constructed. Many, much more subtle, cycles are possible. M and N should not have started the new computation phase, and B and G should not have agreed to changing the edge costs. When a node gets a 'done' signal, it must communicate with its neighbors to establish the costs of incident edges for the next computation phase. We examine the requirements of the protocol that achieves this.

Let A be a node that has got the 'done' signal. Some of A's neighbors may have received the 'done' signal, while others may not have. Let B be A's neighbor, and assume that B has not yet received the signal. When A communicates with B, in an attempt to establish the cost of edge (A,B), B must tell A that it will respond as soon as it gets a 'done' signal. When B gets the 'done' signal, then everything is fine, since both nodes have got the 'done' signal and can change the cost of an edge connecting them. When all edges incident to a node have their cost reestablished and new-branches marked as old-branches, the node can proceed with the new phase, if possible.

If B were to "blindly" agree on the new cost of the edge (A,B), then subsequent actions of A may produce undesirable race conditions, which cause yet another computation phase to be initiated while one is already in progress. We illustrate this in figure 2.7. Assume that the MST has just been constructed and node E broadcasts the 'done' signal.

The new edge costs are also shown. Nodes will only establish them when they get the 'done' signal. Assume that G and A have received the 'done' signal while B has not. A will now establish the cost of edge AB and B will blindly agree. A will then remove branch AF and create AB. Assume that B now gets the 'done' signal. It will forward it to A and A will go about reestablishing edge costs for another computation phase and could easily construct cycles.

Therefore when A communicates with B about the new cost of edge AB, A is kept waiting until B has got the 'done' signal. Only then is the cost of AB reestablished. Hence, when a node has been completely reinitialized, all its neighbors have also got the 'done' signal and agreed on these edge costs. This simple protocol for reinitialization is sufficient to guarantee that a new phase is not initiated while one is currently active even when new nodes are being added. We show this in section 2.6.5. The proof of this is straight forward. When a reinitialized node subsequently becomes a leaf and/or master, it is guaranteed that all its neighbors will process a signal it sends them only after they have been reinitialized too. Hence, a signal belonging to a new phase is not processed by a node until it too is ready for the new phase. Hence, the reinitialization protocol serves to synchronize the knowledge two nodes have of the edge that connects them for the next computation phase. More esoteric reinitialization protocols may be developed.

It is appropriate at this point to discuss the properties of the broadcast routing scheme by which 'done' signals are delivered to all

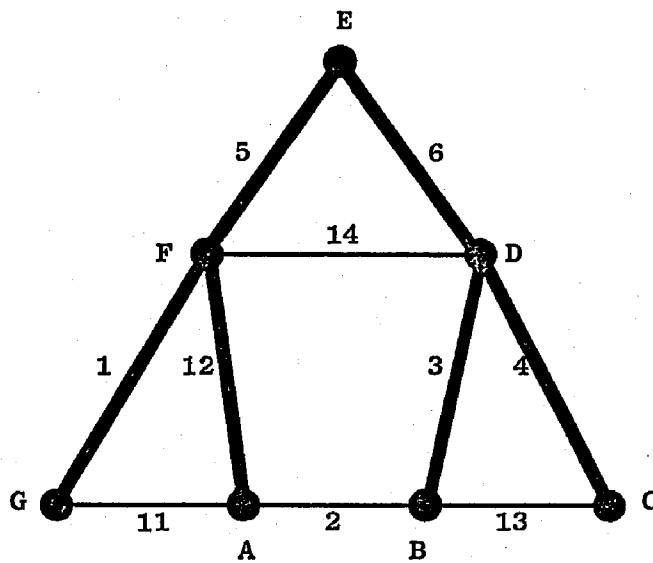


Figure 2.7. UNDESIRABLE RACE CONDITION
IF NODES BLINDLY AGREE ON NEW EDGE
COSTS.

the nodes. The 'done' signal is forwarded along the new-branches of the MST. Since there is no alternate routing strategy, a node cannot get a duplicate copy of a 'done' signal. Further, when a 'done' signal of a particular phase is broadcast, then 'done' signals of the previous phase no longer exist. This is true since all nodes do not complete their reinitialization until they too receive the 'done' signal. The broadcast routing scheme therefore guarantees to deliver exactly one copy of the 'done' signal to a node, provided that edges and nodes do not go down until after all nodes have learned of the termination of the current phase.

When a node enters its reinitialization code because it receives a done signal, it does not process any messages not related to reinitialization until after it is ready for the next phase. Analogously, if edge cost reestablishment messages arrive before the node gets the 'done' signal, then the node tells its neighbor that the response will be coming shortly. Conceptually, one can imagine that two nodes have got two interprocess communication channels between them - one for receiving and transmitting signals related to MST construction and the other for reinitialization. The interprocess communication channels should be reliable and provide sequenced communication [Cerf74, Cerf74a, Sunshine75].

2.6.4 Constructing the MST from Scratch

Let us examine the operation of the algorithm when there is no spanning tree initially overlaid on the nodes of the network. The nodes, however, can all be considered connected to a fake node by an edge. Hence, there is a spanning tree overlaid on the nodes of the network, and all the nodes are leaf nodes.

Hence, assuming that each node knows the identities of its neighbors and the cost of the edges connecting it to its neighbors, the algorithm is no different from that described in section 2.6.2.4 except that the initializations for the first phase have all nodes labelled as leaves, and there are no old-branches.

Notice that this algorithm is now very similar to the static distributed algorithm described in section 2.4.3. The signals used in the adaptive algorithm are 'make this edge (old-branch) into a new-branch' and 'become master'. These correspond, respectively, to 'mark this edge' and 'become master' in the static algorithm. There is no signal in the adaptive algorithm corresponding to 'become master and make this edge an unmarked branch' in the static algorithm. This is so, because in the adaptive algorithm whenever two nodes convert the same edge into a branch one of them unambiguously becomes master. This is not always the case in the static algorithm where one of the nodes could have created a marked branch while the other could have made that edge into an unmarked branch. There is an extra command in the static case in order to transfer more information in a signal. Hence the static

algorithm of section 2.4.3 is a special case of this adaptive algorithm.

The adaptive algorithm will still have only one node that determines that the phase has terminated. Hence, the algorithm can be used to create an MST from scratch and then have it adaptively change. In the next section we examine the case where all nodes do not initially know the identities of their neighbors and the cost of the edges connecting them to their neighbors.

2.6.5 A Very General Environment

We now show how that algorithm as described in section 2.6.2 can be used in a very asynchronous environment, where edge costs are changing, new nodes are being added to the network, and some nodes are being removed. The algorithm is essentially unchanged from that described in section 2.6.2 but the initial conditions and reinitializations are a little more complex.

During the reinitialization process that a node undergoes upon learning that the current computation phase is over, it may discover that some of the edges incident to it have an infinite cost. This implies that the edge can not be used for communication, and so it is not included in the node's reinitialized fragment state. Such edges may have been branches. The node marks them as old-branches in its data structures corresponding to the edge information so that the node can become a leaf and replace this old-branch. Note that it is likely that the nodes on either end of such a dead branch will attempt to replace

it. This could have happened even when edge costs were not infinite and so this is not a special case.

If all edges to a node become infinite then the node and its neighbors assume that the node has been removed from the network, and it will be excluded from the MST being constructed by virtue of the action each neighbor will take.

Now that we know how to account for nodes being removed from the network, let us examine how new nodes and edges can be added to the network. First, consider how new nodes are added. These nodes could be brand new nodes, or nodes that have become completely partitioned off from the network and now have edges of finite cost connecting them to other nodes. Both situations are equivalent. When a new node comes up, it discovers its neighbors using some simple low level protocol, and attempts to establish interprocess communication channels with them. The node now attempts to establish the edge costs so that it can initialize its data structures and proceed with constructing the MST. None of the neighbors will agree on the edge costs until they too are in the (re)initialization process themselves - because they got a 'done' signal, or because they too are new nodes. When a new node has completely initialized its data structures it can proceed. A new node is initialized as a leaf node and proceeds to connect itself to a node by a branch. It is like any other leaf node now.

This simple method of introducing new nodes into the network is consistent with the rules for termination and initiation of phases. A

new node can proceed with MST construction only at the beginning of a phase. This restriction may impose some delay before a new node can proceed with MST construction. We feel that this delay is worth the simplicity of the phase synchronizing scheme.

Next, consider how new edges can be added to the network. If these new edges came from at least one new node, then it will be introduced into a new computation phase at the same time by both the nodes at either end, as we have just seen. Let us now examine how an edge that once had an infinite cost now has a finite cost, i.e. it can be used for communication again. Note that the nodes on either end are not new and have been participating in computation phases, treating this edge as if it did not exist. Either or both the nodes will discover that the edge is available for communication and establish interprocess communication channels between them. The difficult part is introducing this edge with a new finite cost into the same computation phase for both nodes. If we assume that such edges come into existence only during the reinitialization process for both nodes, then either one of them could initiate reestablishment of the edge cost and the edge would be introduced in the new computation phase for both nodes at the same time. This assumption is necessary for the same reason that edges and nodes cannot go down during a computation phase, or that edge costs can not be changed during a computation phase. That is, the algorithm assumes a fixed topology during a computation phase and any change would cause the state information at various nodes to be inconsistent.

The assumption that no changes in topology can occur during a computation phase, and only during the reinitialization period is not unreasonable. The adaptive algorithm will be typically used in an environment where an MST is constructed, and used for a certain period of time and then reconstructed. Therefore each node will be in the reinitialization process for a longish period of time, deciding what the local topology should be like for the next phase. Nodes could be programmed to wait for a certain period of time when they have reinitialized themselves and found out that they are leaves. Hence, one can imagine that the nodes construct the MST, then spend some time reinitializing. Upon reinitialization the leaves of the MST could wait for some time before starting the new phase. Once they have started the phase, nodes that become leaves perform their usual functions. This does not require clocks in different nodes to be synchronized or even have the same period.

2.6.6 The Packet Radio Network Environment

We now describe how the algorithm can be used in the Packet Radio Network [Kahn75, Frank75] which uses centralized routing. Packets are forwarded from a source repeater along the branches of a tree to the station where they get routed either to a host connected to another network for which the station acts like a gateway, or to a user connected to a destination repeater. The tree along which packets are forwarded is rooted at the station and could be a minimum height tree. If a minimum spanning tree connecting the repeaters and station is

equally satisfactory, then this algorithm can be used to construct the MST when the repeaters are dropped from an airplane and the station is already on the ground. The repeaters and station all have the same algorithm executing in them. The algorithm adaptively recomputes the MST as more repeaters land and discover other repeaters. We assume that repeaters do not go down and edge costs do not become infinite (unless repeaters are in the reinitialization process!). We must, however, permit new edges to be introduced into the network at all times and not only during reinitialization. This can easily be done, as we shall see. Notice that the assumptions for the adaptive algorithm to work have not been violated in this real life application!

Assume that an edge (A,B) can go from infinite cost to a finite cost at any time. Either A or B, or both will discover that this edge is available for communication and establish interprocess communication channels between them. Assume now that A enters its reinitialization code because it gets a 'done' signal. A attempts to reestablish the cost of this edge. If B is also in its reinitialization code, then all is fine and the edge will enter the new computation phase, as we have seen in section 2.6.5. However, it is possible that B may alternatively be in one of the two following states:

- (1) B may just have reinitialized itself and proceeded with the new phase (A and B could not communicate when B got its 'done' signal since edge (A,B) did not exist at that time).

(ii) B may not have yet got the 'done' signal.

A and B must synchronize their actions so that (A,B) has the same cost as seen by both of them for all phases. Let us see how this is achieved. A will be told by B that it is not in the reinitialization process and to wait for the response. A can not, however, wait indefinitely because B may have been in a state described in (i) above. A must treat this edge specially. A assumes that B is in state (ii) above and waits for a certain amount of time. If B responds in that time with establishment of the edge cost, all is again fine. If B does not respond, then A aborts this reestablishment and assumes that B was in state (i) and therefore treats the edge as though it had an infinite cost. A's assumption may have been wrong in that it just didn't wait long enough. In that case the situation and process B goes through upon getting its 'done' signal is symmetrical to what we just described. Hence, if timings are not right, then it is likely that A and B will not introduce this new edge into the network for a number of phases. We believe that nodes will be in the reinitialization process for a longish period of time, and that there are stochastic delays and so this synchronization will eventually come about. This synchronization mechanism is very much in the same spirit as the one used by the Internet Transmission Control Program when it sets up an interprocess communication channel across a very unreliable subnet [Cerf74a, Tomlinson74, Dalal74, Dalal75, Sunshine75].

As repeaters land they will discover the world around them. They do not wait for all repeaters in their neighborhood to land since they

do not know how many there will be. A repeater decides that it has enough neighbors and then considers itself initialized and proceeds to construct the MST. It is possible that a small MST will be constructed in the first phase, and this will become larger as new nodes are added in subsequent phases. It is also possible that a number of small MSTs will be constructed, but as more nodes land or the existence of new edges are discovered the small MSTs will connect themselves to one another producing a final MST.

2.6.7 Analysis of the Algorithm

The adaptive algorithm is relatively simple once a new computation phase has been properly initiated. In terms of the abstract parallel algorithm, leaf nodes decide to connect the new fragment they have information about to another fragment by the minimum cost edge. Since all N nodes eventually become leaves and there are only $N-1$ old-branches, one and only one node determines that the computation phase has terminated. This node informs the others by broadcasting a 'done' signal, along the branches of the MST just constructed.

The reinitialization process that a node undergoes upon realizing that the current computation phase is over is very important. The properties of the protocol by which edge costs are established have been described in sections 2.6.3, 2.6.4, and 2.6.5. The algorithm assumes that certain changes in topology, i.e. edges or nodes going down, only occur during reinitialization, in order to keep the topology from changing during a computation phase, and to guarantee that a computation phase properly terminates.

If edge costs are not distinct, then the asynchrony of the construction process may result in cycles, in a manner similar to that described in section 2.4.5.

The factors that influence the complexity of this algorithm are similar to those described for the static algorithm in section 2.4.4.2, and the actual determination of complexity is a subject for future research. Since control is transferred systematically, there is less chance of redundant computation being performed as a result of long communication delays.

The various fragment information gathering schemes discussed in section 2.5.1 could have been used in this algorithm too, instead of passing the fragment state in each signal.

2.6.8 Conclusions

We have described an adaptive distributed algorithm that converts the old spanning tree into an MST. The properties of this algorithm are similar to those of the static algorithm with the additional feature that it is adaptive. The reinitialization process between two computation phases must guarantee that a new computation phase is not initiated while one is currently in progress. The reinitialization protocol we have described must synchronize the knowledge two nodes have of the edge connecting them for the next computation phase. This requirement and the fact that termination of a computation phase is announced by a broadcast along the branches of the MST requires that

edges and nodes can go down only during the reinitialization process. New edges and nodes can be added to the network at any time, and are introduced into the computation phase correctly. This algorithm can be used in the Packet Radio Network to initially configure the radio repeaters when they are dropped out of an airplane.

2.7 Conclusions

We have presented two concurrent and asynchronous algorithms for constructing MSTs in a distributed environment in which there is no one point of control. One of the algorithms is adaptive to changes in topology under certain constraints. We believe that these are the first algorithms of their kind for constructing MSTs. We have described some other alternatives for gathering state information, but their suitability has still to be determined.

For some applications it might be desirable to construct an MST with the minimum diameter. We feel that by using a concurrent, asynchronous algorithm based on Prim's "greedy"* algorithm, it is not possible to guarantee that the MST constructed will be the one with the minimum diameter.

*The use of this term was found in [Kershenbaum74]. It indicates that at every stage in the construction process, if there is more than one nearest neighbor, then any one is chosen. The algorithm, therefore, constructs an MST but does not optimize any other objective function.

CHAPTER 3

BROADCAST ROUTING ALGORITHMS

3.1 Introduction

Broadcast routing is defined as the capability of the communication subnet to deliver a broadcast message from one host to all destination hosts. This is a special case of multi-destination routing, where a message is delivered to more than one destination. We have described in Chapter 1, the need for such a capability from the communication subnet.

The efficiency of the broadcast is greatly dependent on the nature of the particular subnet over which it is attempted. In this chapter, we describe broadcast routing algorithms for packet switched, store-and-forward computer networks. The ARPANET [Roberts72, McQuillan72] will be used as the model for such PSNs. PSNs have storage and a (small) holding time at every switching node, and so can be thought of as providing statistical time division multiplexed communication.

There are many ways of performing broadcast in PSNs so as to reduce the total amount of communication needed, thereby performing the broadcast quickly and cheaply, as well as lowering the possibility of subnet congestion. We describe the algorithms and show the qualitative tradeoffs. Much of this discussion is based on a private communication from Cerf [Cerf76]. In Chapter 4, we determine lower bounds on the

performance measures for the algorithms, in order to compare them quantitatively. The two important measures of performance are the number of packet copies generated and transmitted to broadcast a packet to all nodes, in packet-hops, and the delay in propagating the packet to all nodes.

Sections 3.2 to 3.7 describe the various algorithms, section 3.8 discusses the reliability of broadcast protocols and section 3.9 discusses the tradeoffs between global and subgroup broadcast routing algorithms.

3.2 Separately Addressed Packets

Conventional packet switched networks based on point to point circuits, like the ARPANET or Telenet, use fixed or dynamic routing tables to minimize the number of transmissions (store-and-forward) required to move a packet from source to destination. Such a system is designed to support efficient point-to-point communication. Broadcast communication can be achieved from such a system by sending a distinctly addressed copy of the packet to each destination. There are several drawbacks of such a scheme.

(i) A larger number of packet copies are forwarded and transmitted for every broadcast than is necessary. We show later that the minimum number of packet-hops to broadcast a packet in a subnet with N nodes is $N-1$.

(ii) As a consequence of more packets being transmitted than necessary, the level of congestion within the subnet may increase, thereby increasing the delay for delivering packets.

(iii) In some communication subnets, a virtual circuit may have to be set up between the source and each destination in order to transmit each broadcast packet. This is an unnecessary waste of resources since these virtual circuits may not be used for further communication.

We now examine algorithms that take advantage of the basic store-and-forward nature of such networks. In order to minimize the

delay to propagate a broadcast packet, it should be forwarded along the shortest paths from the source to the destinations. Figure 3.1a and 3.1b show the shortest path trees from nodes 5 and 3 respectively. Note that these shortest path trees are not unique in this example. Since all communication links, in this example, have the same cost, the shortest paths are based on hop-count. The broadcast packet is forwarded from the source along such a tree and delivered to the nodes that lie on the path. In a subnet with N nodes, each source has $N-1$ destinations. The shortest path tree connects the source to all the other nodes by $N-1$ branches, and therefore the minimum number of packet-hops to broadcast a packet is $N-1$.

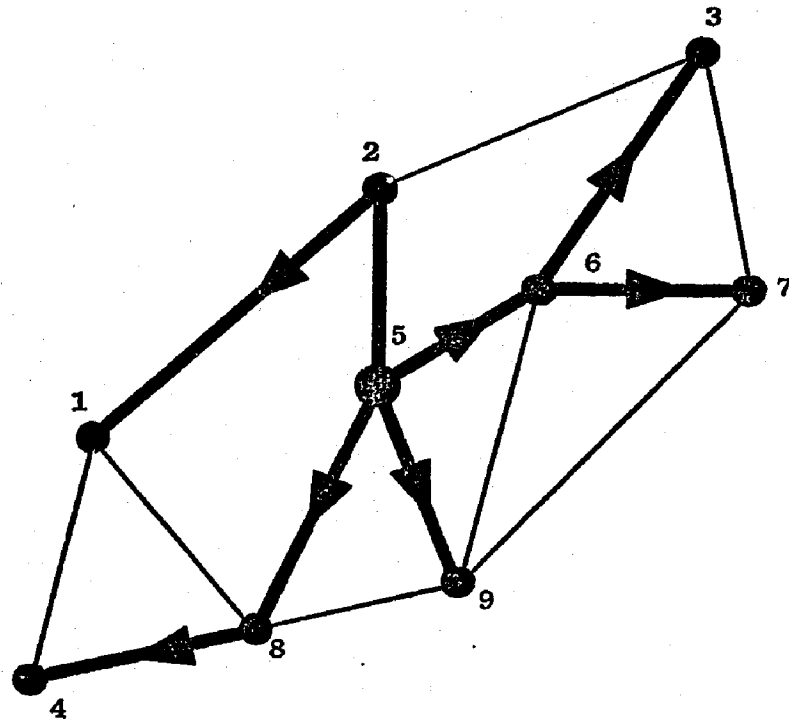


Figure 3.1a. SHORTEST PATH TREE FROM NODE 5.

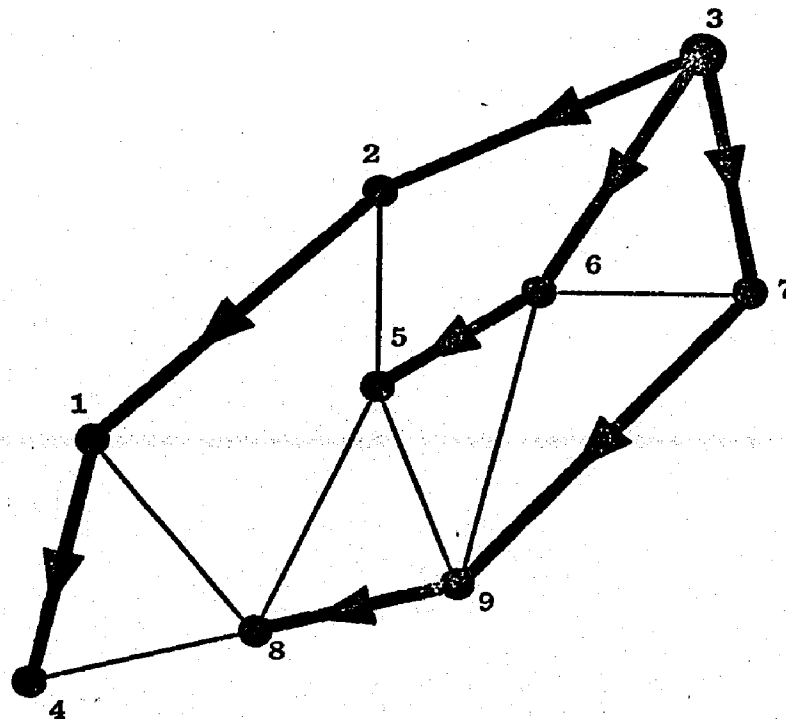


Figure 3.1b. SHORTEST PATH TREE FROM NODE 3.

3.3 Multi-Destination Addressing

If a multi-destination addressing scheme is available within the subnet, then it can be coupled with the existing subnet routing algorithms to minimize the delay and number of packet copies transmitted to deliver a broadcast message to all destinations. This is because the shortest path routing algorithms of the network have information isomorphic to the N shortest path trees discussed above.

The basic problem, even with the multi-destination addressing scheme is for each node to decide where to forward the multi-addressed packet. For example, in Figure 3.2, a source at node 5 might send a multi-addressed packet to destinations connected to nodes 3, 6, 7, via link "a" (as seen by node 5). A simple selection criterion for labelling the multi-destination packets is to assign destinations according to the preferred route as indicated in the routing table. On arrival at node 6, copies of the packet labelled (3,6,7)* would be forwarded to nodes 3 and 7, relabelled with destination 3 on link a and destination 7 on link b (as seen by node 6).

At their origin, in general, copies of broadcast messages are assigned multiple destinations according to the routing table at the origin node. The addresses of all destinations requiring routing out link "a", for example, would be attached to a single copy of the message sent on link a. The next node would create more copies, if necessary, assigning each copy a partition of the incoming address list.

*Assume that host addressing is synonymous with node addressing for this simple example.

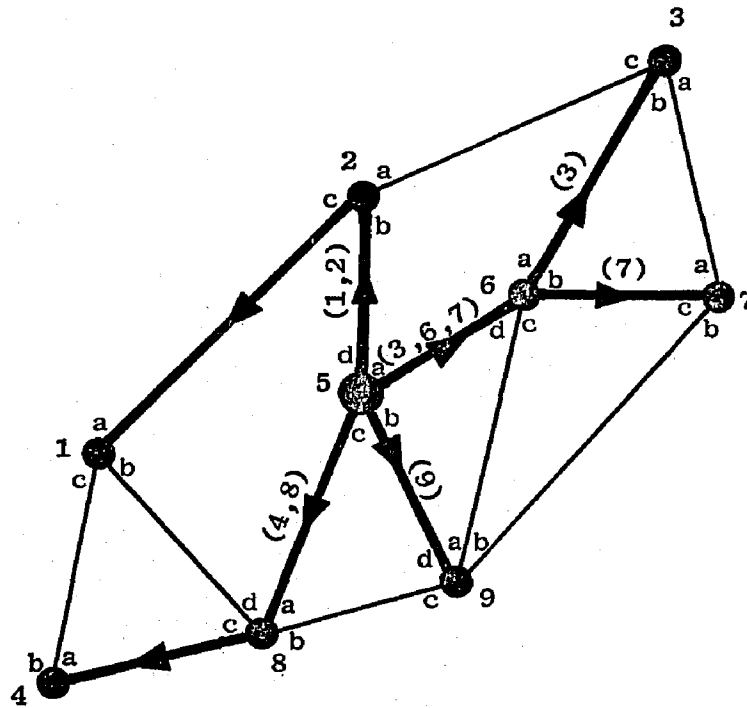


Figure 3.2. MULTI-DESTINATION ADDRESSING
ALONG SHORTEST PATHS FROM NODE 5.

Network flooding (endless retransmission and forwarding of a broadcast packet) is prevented because messages which arrive at a destination node no longer carry that address in any subsequent copies.

Using such a capability, packets are forwarded and delivered along the shortest paths from the broadcaster to the receivers. In a subnet with N nodes, $N-1$ packets are transmitted. The time for completion of the broadcast is equal to the delay to the receiver furthest away.

Routing algorithms that may be used in PSNs [McQuillan74] would then remain unchanged, thereby permitting the system to adapt easily to changing network conditions. The headers of packets exchanged between switching nodes would have to be designed to carry multiple destination information. This could be done by having a variable length packet header, or having a fixed length bit map indicate the various destinations. Alternatively, there might be a fixed length header capable of carrying a restricted number of multiple destinations. The disadvantage of such a scheme is that at the originating node more than one copy of the packet may be transmitted on the same link, if the number of multiple destinations optimally reachable on that link is larger than the number of multiple destinations the packet header can carry. This may lead to more packet copies being transmitted by the other nodes as well. The "badness" of this scheme is the number of packet copies greater than $N-1$ transmitted to do broadcast. Separately addressed packets is a special case of restricted multi-destination addressing where the number of addresses is one. The forwarding function of the switching node would have to be sensitive to the

multi-destination addresses, and therefore is more complex to implement.

If the underlying routing algorithm changes the shortest path tree after the broadcast packet has left the source node, then it is possible that the addressing decision made by the source or intermediary forwarding nodes will be suboptimal. Therefore, more packets copies will be transmitted than necessary, thereby also increasing the delay. Hosts will, however, not get duplicates because packets contain destination addresses, and at any time there is only one packet in transit with a particular destination address.

The big advantage of this scheme is that it permits broadcast to a subset of all possible destinations.

Many subnets may not wish to have a variable length address field, or a bit map encoded multiple destination address, but would rather have a special address "all" signifying that the packet is to go to all destinations. Packets always have the source host in the header of the packet. To avoid endless retransmission of a broadcast packet, without the use of multi-destination addressing, it is necessary for each node to know whether it lies on the current shortest path from a source to a destination. That is, knowing the source of a broadcast packet, a node must decide over which links to forward copies. It obviously need not forward copies back on the link the packet arrived on. We now examine broadcast routing algorithms based on this constraint.

3.4 Hot Potato Forwarding

We could use the "hot potato" propagation scheme first proposed by Baran, et al, [Baran64] to achieve broadcast. Each node copies arriving packets, irrespective of their source, to all outgoing links except the one on which it arrived. The scheme, as it stands, produces network flooding very quickly. A scheme for discarding "old" packets must be used to avoid network flooding.

The simplest scheme for detecting old packets is to have sequence numbers assigned to broadcast packets transmitted from a source. Each node checks to see if the broadcast packet that arrives is an old duplicate. If it is, then the packet is discarded. If it is a new packet, then the sequence number is remembered and the packet forwarded along all the other outgoing links. Since nodes have a finite memory space, sequence numbers must be purged from the memory after a suitable time. In order that this scheme work, packets must have an upper bound on their lifetime in the subnet, and sequence numbers must not cycle within this lifetime. This lifetime determines the lower bound on the time that a node must remember a sequence number.

Alternatively, if a "hop count" were kept in each packet, a packet could be discarded by a node if the hop count exceeded the longest path in the network (usually $N-1$ in a network with N nodes). With some probability, this threshold could be reduced to a smaller number without affecting the success of the broadcast; of course the threshold must be larger than the diameter of the network. Consider what happens in a hot

potato broadcast from node 5 in figure 3.3. Let the threshold be $N-1=8$ (it could be set to the maximum hop distance as currently known to the source node. Under changing conditions this heuristic will fail, but perhaps not so badly that it is useless).

For convenience, let us represent the initial copies of broadcast packets as P_i where i ranges over the number of copies made by the source node. As these packets are propagated, let us represent the propagated copies as $P_i(j)$ where j is the hop number. Thus, on the first hop, $P_1(1)$ is copy 1 arriving at the first node after departure from the origin. Likewise for $P_2(1)$ etc. Figure 3.3 illustrates the resulting packet propagation for 3 hops. Table 3.1 shows which packets have arrived at which nodes after each hop. It is clear that after 3 hops, some nodes have already received 5 copies of the original packet, and in fact one node has received 7!*

Obviously, the threshold choice is highly critical. Furthermore, duplicate detection will require that broadcast messages contain sequence numbers which will not cycle during the lifetime of a broadcast propagation.

*Nota Bene: All $P_i(j)$ are identical in content. The objective is to deliver at least one $P_i(j)$ to each node, and to deliver as few duplicates as possible.

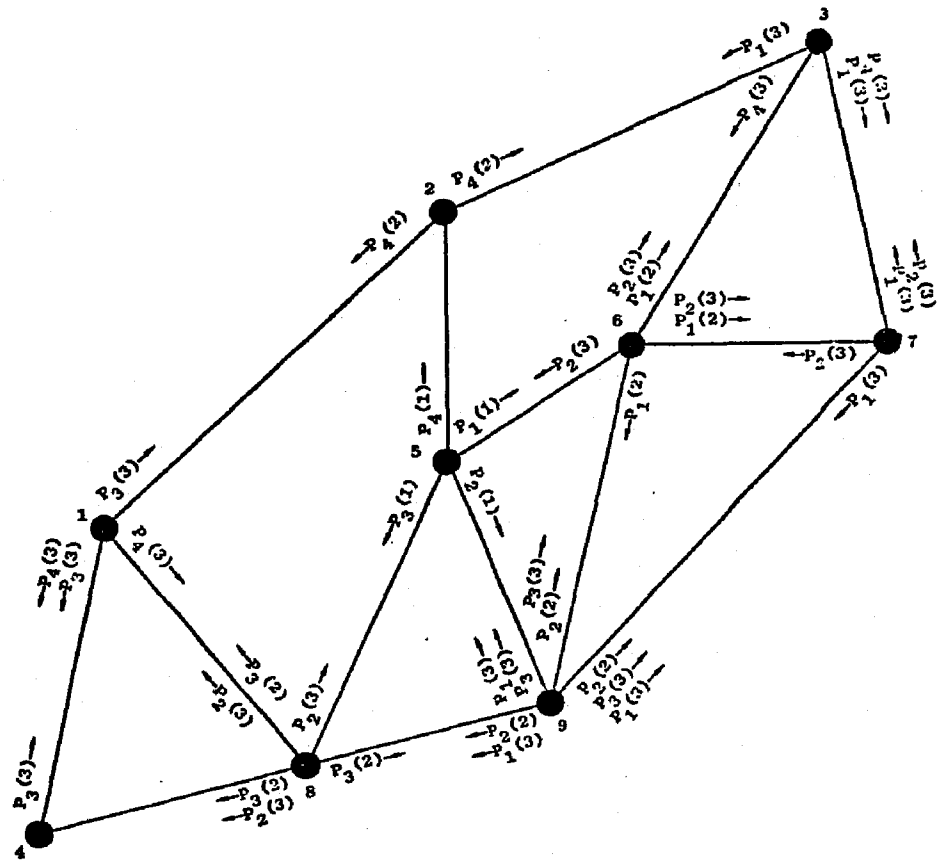


Figure 3.3. PACKET PROPAGATION USING HOT POTATO FORWARDING AFTER 3 HOPS.

Table 3.1

HISTORY OF PACKET ARRIVALS FOR HOT POTATO PROPAGATION

Node No.	1	2	3	4	5	6	7	8	9
Hop 1	X	P ₄ (1)	X	X	X	P ₁ (1)	X	P ₃ (1)	P ₂ (1)
Hop 2	P ₄ (2) P ₃ (2)	X	P ₄ (2) P ₁ (2)	P ₃ (2)	X	P ₂ (2)	P ₁ (2) P ₂ (2)	P ₂ (2)	P ₁ (2) P ₃ (2)
Hop 3	P ₃ (3) P ₂ (3)	P ₃ (3) P ₁ (3)	P ₂ (3) P ₁ (3) P ₂ (3)	P ₄ (3) P ₃ (3) P ₂ (3)	P ₂ (3) P ₂ (3) P ₁ (3) P ₃ (2)	P ₄ (3) P ₂ (3) P ₃ (3)	P ₄ (3) P ₁ (3) P ₂ (3) P ₁ (3) P ₃ (3)	P ₄ (3) P ₁ (3)	P ₁ (3)

3.5 Source Based Forwarding

If the N optimal shortest path trees (one for each node as the source) can somehow be incorporated in each of the nodes, then broadcast packets to be sent to "all" destinations can be forwarded based on their source. For each source, each node knows along which links incident to it, it must forward the packets. Table 3.2 illustrates a broadcast forwarding table derived from a fixed, minimum hop routing strategy for the network in figure 3.2. Each column of Table 3.2 is a broadcast strategy operated by the node whose name appears at the top of the column. Each row represents the spanning tree used to broadcast packets from the source at the left of the row. If node i receives a broadcast packet whose source is j , it looks at the j^{th} row of its routing column and sends copies of the packet out along the indicated links. For instance, node 6 will send copies of a packet originating at node 5 out links a and b (as seen by 6). If the table entry is empty, then the packet is not forwarded further. Each node will have one column of this forwarding table.

There is no network flooding, or duplicate detection in this case either. This scheme also tends to minimize propagation delay and the number of packets transmitted. The disadvantage is that it requires additional table space at each node, and that in its current form the algorithm does not adapt to changing network conditions. In the next section we describe a similar algorithm which is adaptive to changing network conditions.

Table 3.2

BROADCAST FORWARDING TABLE FOR NETWORK SHOWN IN FIGURE 3.2
 (Note: Assume host addressing is synonymous with node addressing)

		Forwarding Node								
		1	2	3	4	5	6	7	8	9
Source Node	1	a,b,c	a	b					a,b	b
	2	c	a,b,c	a		a,b,c				
	3	c	c	a,b,c			d	b		c
	4	a	a		a,b	a			a,b	b
	5		c			a,b,c,d	a,b		c	
	6		c	c		c	a,b,c,d		c	
	7			c				a,b,c	c,d	c,d
	8					a,d		a	a,b,c,d	b
	9					d	a		c,d	a,b,c,d

3.6 Reverse Path Forwarding

We now describe another broadcast routing algorithm that forwards packets based on their source of broadcast. The simple version of the algorithm does not use a forwarding table like Table 3.2, but only the routing table that a node would normally use to route packets. This simple scheme is suboptimal in the number of packet copies transmitted, and so we extend it to use a table similar to Table 3.2. This table is constructed dynamically and therefore the algorithm is adaptive to changing network conditions. Note that even the suboptimal algorithm would be adaptive since it makes use of the routing table, which in most communication networks is dynamically updated. We will describe the algorithm in context of the routing algorithm and tables used by the IMPs in the ARPANET. Appendix C briefly reviews this routing algorithm.

The idea behind the algorithm was first proposed by Metcalfe (in a private communication to the author). The broadcast packets are not forwarded along the tree of shortest paths that connect the source to the destinations, but rather the tree of shortest paths that connect the destinations to the source. Hence, the name reverse path forwarding. In the event that communication costs in either direction of an edge are same, the two trees will be identical.

3.6.1 The Simple Scheme

When a broadcast packet from a particular source (with destination "all") arrives on a particular communication link, the node determines

from its routing table whether it would route a packet to that source along the same link. If so, the packet is delivered to all hosts connected to the node, and forwarded along all links except the one on which it arrived. The node does this because it concludes that it lies on the shortest path that connects some of the destinations to the source. If the packet did not arrive on the correct link, then it is discarded.

This simple algorithm guarantees prevention of network flooding, because the paths along which packets are accepted for delivery and forwarding are cycle free. We show why this is so. A packet is accepted on a link if the link is part of the paths that connect the destinations to the source by the shortest paths as determined by the normal routing algorithm. If the normal routing algorithm creates, at any time, a unique route between two nodes, then the shortest paths from the destinations to the source will be a tree that is cycle free. These paths will also be cycle free if the normal routing algorithm creates cycle free routes between any two nodes. Routing algorithms, in general, satisfy both these properties.

This scheme requires no modification to the packet format currently used in the ARPANET. The IMP code would, however, have to be enhanced to perform the forwarding function. Abstractly, in an ALGOL-like language, the program in the IMP would be:

```
if PACKET.DEST = ALL
then begin
    if INCOMING_LINK = ROUTING_TABLE(PACKET.SOURCE)
    then [deliver to all hosts, and forward along all other edges]
    else [discard the packet]
    end
else begin
    [route this normal packet];
    end;
```

The number of packets transmitted is, however, larger than the minimum. The delay for propagating the packets may be larger than the optimal value if the reverse path tree is very much different from the shortest path tree. Figure 3.4 illustrates how packets would be forwarded along the reverse paths from node 8. The notation for labelling packets is identical to that used in section 3.4. Note, again, that the packets that are accepted by the nodes for delivery to the hosts connected to them, arrive on branches of a tree that connect the destinations to the source by the shortest path.

3.6.2 The Optimal Scheme

Notice from figure 3.4 that the total number of packet copies transmitted is 22, while the minimum number is $N-1=8$. The total number is equal to the sum of the connectivities of each node minus $(N-1)$, since each node forwards a packet along all but one link incident to it, except for the source which transmits the packet on all links incident to it. For most network topologies this number is much larger than the minimum.

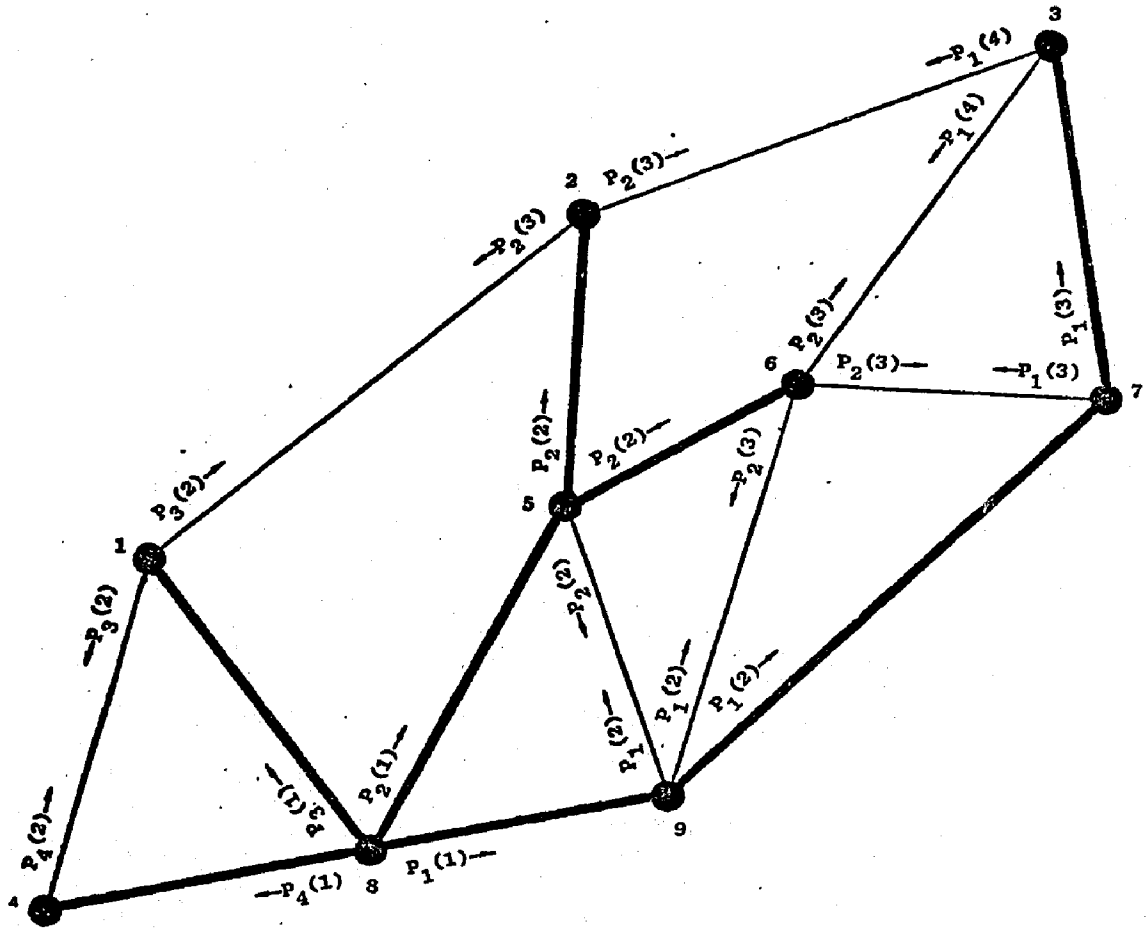


Figure 3.4. SUBOPTIMAL REVERSE PATH FORWARDING INITIATED FROM NODE 8.
 (Note: host addressing synonymous with node addressing.)

If each node knew along which links to forward the packet (as determined by columns of Table 3.2), then the number of packet copies transmitted would be equal to $N-1$. Such a BROADCAST_FORWARDING_TABLE at node 6, of the network in figure 3.2, is illustrated in figure 3.5

In terms of the reverse path forwarding model, in order to construct such a forwarding table, a node must know to which destinations a given link will be used by its neighboring nodes to transmit a packet. Hence, if a broadcast packet arrived from one of those "destination", then the node will know along which links to forward it. Periodically, each node will transmit to its neighbors a LINK_USAGE_TABLE indicating to which destinations the node will use this link for transmitting packets. When a node receives such a table, it is written over the appropriate column of its BROADCAST_ROUTING_TABLE. The row to be written over is the row corresponding to the link that the arriving LINK_USAGE_TABLE came in on. The LINK_USAGE_TABLE can easily be derived from the ROUTING_TABLE at a node. The LINK_USAGE_TABLES are the inverse of the ROUTING_TABLE, i.e. they indicate for which destinations a particular link will be used. Figure 3.5 also illustrates the ROUTING_TABLE from which the LINK_USAGE_TABLE for link d was derived. This LINK_USAGE_TABLE will be transmitted to node 5.

The LINK_USAGE_TABLE could be transmitted every $2/3$ of a second along with the MINIMUM_DELAY_TABLE. The BROADCAST_ROUTING_TABLE and LINK_USAGE_TABLES could be stored as bit maps since their entries are logical values.

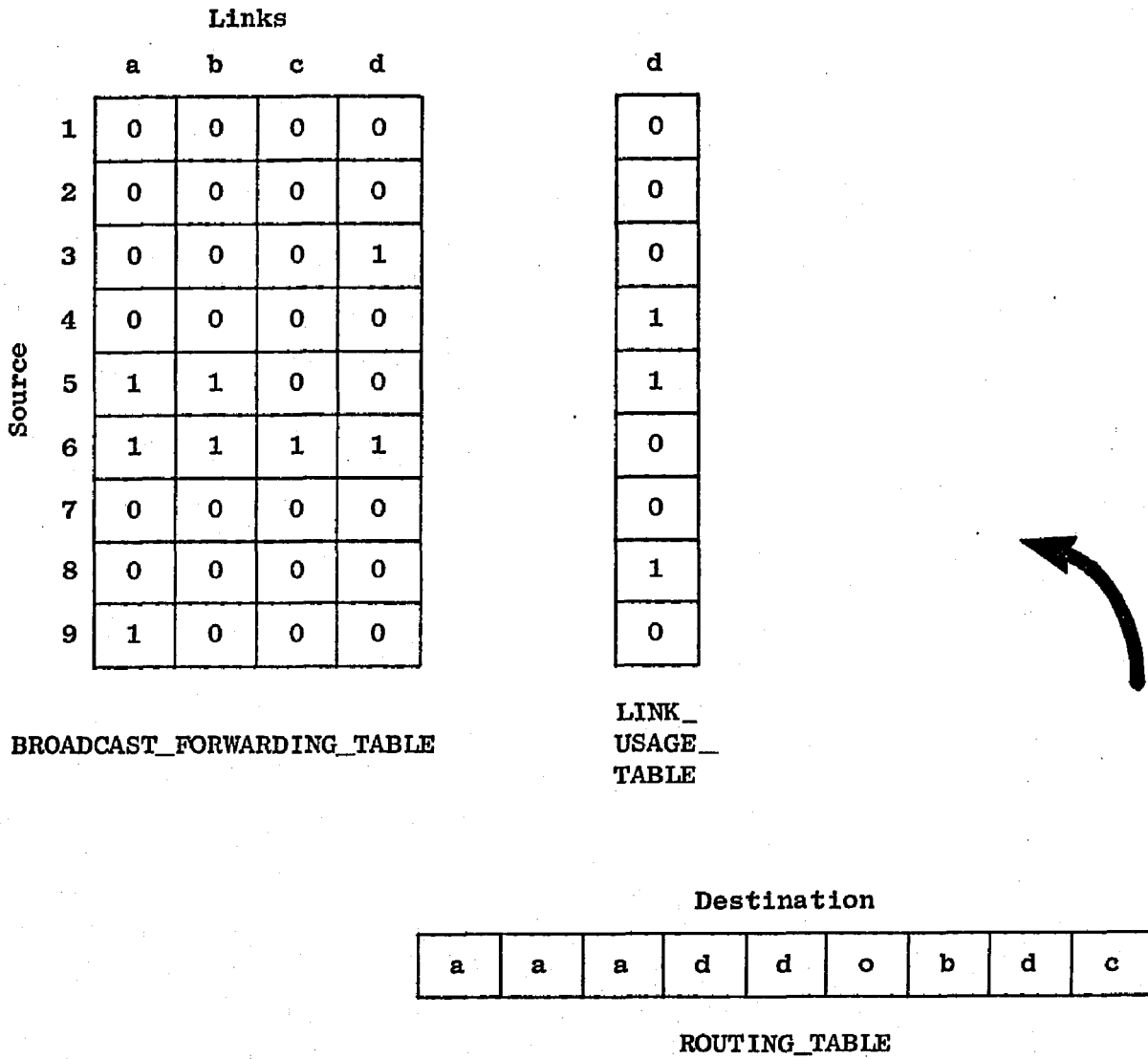


Figure 3.5. THE TABLES AT NODE 6 FOR THE NETWORK OF FIGURE 3.2 TO ACHIEVE OPTIMAL REVERSE PATH FORWARDING.

Note that the `BROADCAST_FORWARDING_TABLE` is a function of the source of the broadcast message. It is very easy to show by an example that if the table was indexed by link numbers instead, then the scheme does not guarantee that the minimum number of packet copies will be transmitted. This optimal scheme is an adaptive version of the source based forwarding algorithm.

Abstractly, in an ALGOL-like language, the forwarding function of the IMP could be expressed as:

```
if PACKET.DEST = ALL
then begin
    if INCOMING_LINK = ROUTING_TABLE(PACKET.SOURCE)
    then begin
        [deliver the packet to all hosts];
        [forward the packet along links determined by the row
        of BROADCAST_FORWARDING_TABLE(PACKET.SOURCE)];
    end
    else [discard the packet]
    end
else begin
    [route this normal packet];
end;
```

3.6.3 Reliability Issues

We now examine what happens if the adaptive routing algorithm changes the reverse path tree from a particular source, once a broadcast packet has been transmitted from the source, but has not been received by all destinations. We will show that the reverse path forwarding algorithm can not guarantee to even deliver at least one copy of the broadcast packet to each destination.

In the simple version of the algorithm, a broadcast packet is transmitted along every link of the network. At any instant a node always has one and only one branch corresponding to a given reverse path tree. The reverse path tree for a given source may, however, change during a broadcast, thus causing a duplicate packet to be delivered to a host. For example, in figure 3.4, assume that node 2 has just received packet $P_2(2)$ along link (5,2) which is a branch, but $P_3(2)$ from node 1 has not yet arrived at 2. Node 2 will deliver the packet to the hosts connected to it, and forward the packet appropriately. The ROUTING_TABLE at 2 may now be updated, such that the appropriate link to use to get to node 8 is (2,1) and not (2,5). Therefore, when $P_3(2)$ arrives from 1, it too will be delivered to all hosts connected to 2.

Similarly, if $P_1(4)$ and $P_3(2)$ arrived at 2 before $P_2(2)$, they would be discarded as (2,5) is the appropriate link to get to node 8. The ROUTING_TABLE at 2 may now be updated to replace (2,5) with (2,1). Subsequently, when $P_2(2)$ arrives it will be discarded and node 2 will not have received a single copy of the packet.

Now, let us examine the optimal scheme in which packets only traverse the paths of the reverse path tree. In figure 3.4, assume that $P_2(2)$ has arrived at node 6 from 5. It will be delivered to all hosts connected to 5 and not forwarded any further. At this time assume that $P_1(2)$ has not yet arrived at node 7 from 9. The update algorithm in the meantime replaces the branch (7,3) by (6,3). Hence, when $P_1(2)$ arrives at node 7, it will be delivered to all hosts connected to 7, but will not be forwarded. As a consequence, hosts connected to node 3 will never receive the broadcast packet.

Similarly, if $P_1(2)$ and $P_1(3)$ had arrived respectively at nodes 7 and 3 before $P_2(2)$ reached node 6, and then branch (7,3) was replaced by (6,3), hosts connected to 3 will receive duplicate copies of the broadcast packet.

Note that the optimal scheme forwards broadcast packets only if they arrived on the "correct link". That is, the same link that the node would have used to transmit a packet to the source of the broadcast packet. This was necessary in the simple scheme to prevent network flooding. In the optimal scheme the test is not necessary (if the reverse path tree did not dynamically change), because packets would only be forwarded along branches of the reverse path tree and therefore only arrive on the "correct link". If the reverse path tree dynamically changes during a broadcast, then a node may forward a packet along a link it thinks is a branch, but its neighbor at the other end thinks is not a branch. For example, in figure 3.4, node 7 may forward $P_1(3)$ to 3, while 3 has in the meantime sent 7 a LINK_USAGE_TABLE that does not include link (3,7) in the reverse path tree for source 8. Hence, node 3 will conclude that $P_1(3)$ did not arrive on the "correct link" and will discard it. This may result in 3 never receiving a copy of the packet. Alternatively, if nodes accepted and forwarded broadcast packets based on their source irrespective of the link they arrived on, then more packets would be transmitted than necessary and some nodes may receive duplicates.

Hence, hosts may get duplicates or not receive a packet. We examine the consequence of such situations in section 3.8.

3.7 Forwarding along a Spanning Tree

If a single spanning tree was embedded on the existing subnet topology, then any node on ~~the~~ spanning tree could initiate a broadcast, and the packets would be forwarded along this tree to all destinations. Each node knows which of the links incident to it are branches of the spanning tree. Hence, a broadcast packet arriving on one such branch would be delivered to all hosts connected to the node, and forwarded along the remaining branches. Note that the forwarding function is independent of the source of the broadcast. Such a technique results in the minimum transmission of packets, $N-1$ in a subnet with N nodes. The time for completing the broadcast is a function of where it was initiated, as in some cases, some of the transmissions could take place concurrently. The worst case time for completing the broadcast is a function of the diameter of the spanning tree. Hence, the delay to propagate a packet to all destinations depends on the particular tree chosen to span all the nodes.

We have chosen to use the minimal spanning tree. This is because we have devised distributed algorithms for constructing MSTs in computer networks (see Chapter 2). Hence, if the load conditions in the network were to change it is possible to adaptively construct a new MST in parallel. Figure 3.6 shows the communication subnet of a PSN with the embedded minimal spanning tree. If broadcast was initiated from a host connected to node 6, then a packet would be transmitted along each of the minimal spanning tree branches in the directions shown in the figure. This technique assumes, of course, that the cost of

communication on a branch of the minimal spanning tree is same in both directions. This is not true, in general, for PSNs, since the traffic patterns determine the queueing delays in either direction of the link. It is, however, not a bad approximation.

It might be argued that if all hosts broadcast very often, then the links comprising the minimal spanning tree would become very congested. We know that for a small number of broadcasts such a technique is preferable, and feel that even for a large number of broadcasts it may still be suitable. This feeling is based on the fact that if there were no special broadcast routing scheme, then by transmitting a separate packet to each destination, far more congestion would be introduced. Of course, if the minimal spanning tree were able to reconfigure itself dynamically to changing load conditions then such a technique is far more suitable.

The minimal spanning tree routing scheme is very simple to implement, does not introduce flooding, and minimizes the number of packets transmitted. It does not, however, guarantee to minimize broadcast delay. The MSTs constructed by the distributed algorithms are not guaranteed to be minimum diameter MSTs either. This scheme does not adapt easily to changing network conditions, since it is a reasonably complex task to reconstruct the MST. Further, the old MST must be used for forwarding broadcast packets until the new one is constructed, or else all destinations may not be reached. Synchronizing the switch over from one MST to the other may be a little complex too.

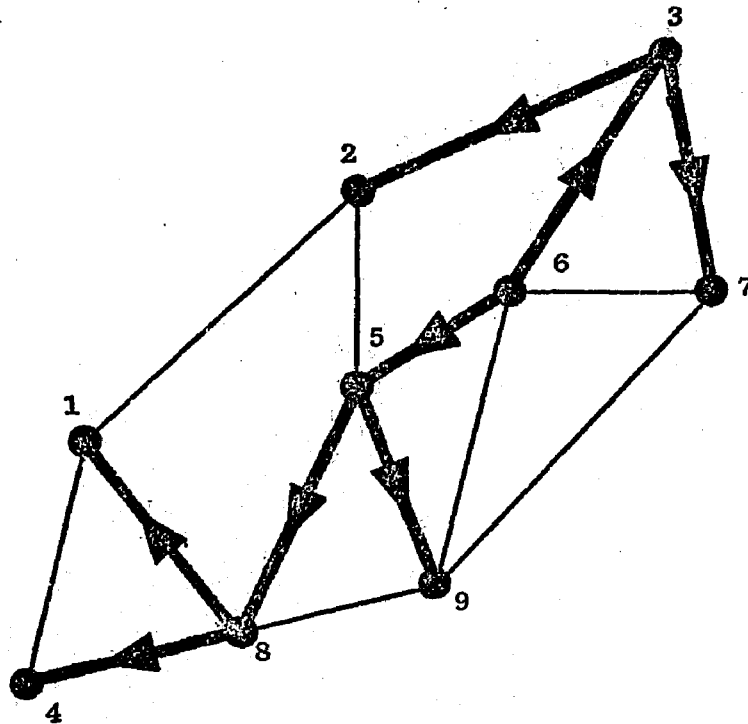


Figure 3.6. BROADCAST ALONG THE MST INITIATED FROM NODE 6. (Note: edge costs are not equal in this example.)

3.8 Reliability of Broadcast Routing Protocols

An interesting issue is whether broadcast routing within the subnet can be made reliable, and what the implications of this is on the higher level broadcast protocols in the host.

We have seen in the previous sections that under the assumptions of a perfectly reliable communication subnet, some of the broadcast routing algorithms are reliable. Separately addressed packets and multi-destination addressing guarantee to deliver one and only one copy of a broadcast packet to all destinations even if adaptive routing algorithms are used within the subnet. Source based forwarding and MST forwarding also guarantee to deliver exactly one copy of a message to each destination because they use a fixed routing strategy.

Hot potato forwarding guarantee to deliver at least one copy of the broadcast packet to each destination. The reverse path forwarding scheme, however, may deliver duplicates or even no copy. This happens because a complete reverse path tree may not exist at all times when the adaptive routing algorithm is in the process of changing it.

In reality, the subnet is not perfectly reliable and so it is likely that any of the schemes may deliver duplicates or no copy of the broadcast packet. The important issues are to determine what high level operations should be performed within the subnet or the hosts, in order to guarantee an acceptable level of reliability. This depends, of course, on the degree of reliability required by the applications that make use of this capability.

In some data gathering systems the real time nature of the system may make it nonsensical to retransmit broadcast packets from the source, because the time value of the information drops too rapidly. In a voice conferencing system, for example, the delays introduced by an attempt to retransmit digital voice packets may be less tolerable than the loss of a broadcast packet or two.

In some applications one might take the position that unreliable broadcast is a prelude to reliable point to point communication; for example when a TIP in the ARPANET locates an RSEXEC server [Cosell75].

There will be other applications in which one and only one copy of the broadcast packet must be delivered to all hosts. The broadcast protocol within the hosts or the subnet (should it guarantee this) must sequence broadcast messages (packets) generated from the source, so that duplicates may be filtered at the destinations. Further, responses to a particular broadcast can contain the sequence number of the broadcast to identify with which message the response is associated.

It may be necessary for the recipients to acknowledge the broadcast messages (packet), so that the source may retransmit only to hosts that did not respond, thus reducing the total bandwidth used in the reliable broadcast. We believe that any kind of acknowledgement scheme used must be between the destinations and the source. Schemes in which the subnet attempts to collect and merge, in a distributed fashion, the acknowledgements into one acknowledgement are not preferable. For example, attempts at merging the acknowledgements along a tree to the

source may deadlock or be unreliable because of timing problems. Such problems do not arise during broadcast because there is no need to wait before transmitting, say two copies of the packet.

We have not described in any detail the structure and properties of a high level reliable broadcast protocol analogous to reliable interprocess communication protocols [Cerf74a, Cef74b, Sunshine75] if implemented within the hosts, or virtual circuits [Rybcznski76] if implemented within the subnet. Clearly there are many problems associated with port addressing, sequencing, duplicate detection, and guaranteeing delivery to all. Broadcast protocols at the higher level will need to perform these functions even if the subnet is inherently broadcast in nature. Requirements for such protocols will become clearer as distributed computing environments make greater use of such capabilities. This is a very important subject for future research.

3.9 Global vs Subgroup Broadcast Routing Algorithms

We conjecture that it is useful to divide broadcast routing algorithms into two classes, one oriented towards sending a message to (almost) every host on the network, and the other oriented towards sending a message to a relatively small percentage of all hosts. This issue was first raised by Steve Crocker. We use the term global and subgroup to refer to the two classes respectively.

Algorithms of the first class will have a reasonable cost for transmitting a message to all hosts, but essentially the same cost for sending the message to even a small group. Those of the second class will have a reasonable cost for sending a message to a small group, but extravagant cost for sending the message to all hosts. Examples of the first kind are source based forwarding, MST forwarding, hot potato forwarding, and reverse path forwarding, while separately addressed packets and multi-destination addressing are examples of the second.

If algorithms that support efficient global broadcast routing are used to deliver messages to a subset of the destinations, then a number of hosts will receive packets that they will subsequently discard. This is a wasteful use of a critical resource - the communication channel connecting the host to the subnet. In such cases the channel may become the bottleneck.

Of course, techniques of the first class can be modified to be used for subgroup routing. For example, different MSTs could be used for different subgroups. Alternatively, a node that is on a single MST, but

does not have hosts belonging to the subgroup, will forward the packet without delivering it to the hosts. Such adaptations are not very general and require special setting up in the subnet. They can, however, be useful if communication networks wish to provide such special services.

The basis for dividing broadcast routing algorithms into the two groups is based on what we imagine communication networks will be used for. The most usual use of broadcast protocols will be to reach a small fraction of the hosts, and therefore there is the need for efficient subgroup routing algorithms. However, for limited purpose networks, it is easier to imagine the need for communication with all hosts, and therefore it will be desirable to use global broadcast routing algorithms. We feel that global broadcast routing algorithms have very little use in large, public, multi-purpose networks.

There is most likely different reliability attainable from the two classes. Subgroup broadcast protocols may be capable of returning verification of receipt with a reasonable cost, while it would be very expensive to do this for global broadcast protocols in large networks.

Hence, there is an equally important need to consider subgroup broadcast routing algorithms in large computer communication networks.

3.10 Conclusions

We have described five alternatives to separately addressed packets to achieve broadcast communication from store-and-forward packet switched computer communication networks. The properties of the algorithms are summarized below. In the next chapter we determine quantitative measures of performance for these algorithms.

(a) Multi-Destination Addressing

Advantages:

- (i) Adapts easily to changing network conditions.
- (ii) Permits broadcast to a subset of all possible destinations.
- (iii) Tends to minimize delay and number of packets transmitted per broadcast.
- (iv) No flooding or duplicate detection problem.

Disadvantages:

- (i) Requires variable length packet headers, or a fixed length bit map capable of carrying all destination addresses if the minimum number of packet copies is to be transmitted. Restricted multi-destination addressing can be used with a possible increase in the number of packet copies transmitted.
- (ii) More complex to implement since the forwarding function of the node has to modify the header and be sensitive to the multiple destinations.

(b) Hot Potato Forwarding

Advantages:

(i) Very simple to implement in its primitive form.

Disadvantages:

(i) Invites flooding and duplicate detection problems.

(ii) High overhead in superfluous packet copies.

(c) Source Based Forwarding

Advantages:

(i) Tends to minimize propagation delay and number of packet copies transmitted.

(ii) Does not introduce flooding or duplicate detection.

(ii) Does not have the overhead of multiple addressing.

Disadvantages:

(i) Requires additional table space in each node.

(ii) Not easily adapted to changing network conditions.

(d) Reverse Path Forwarding

Advantages:

- (i) Adapts easily to changing network conditions.
- (ii) Does not introduce flooding.
- (iii) Does not have the overhead of multiple addressing.
- (iv) Minimizes the number of packet copies if the optimal version is used.
- (v) Minimizes the propagation delay if the reverse path tree is identical to the shortest path tree.

Disadvantages:

- (i) Requires additional table space if the optimal version is used.
- (ii) More complex to maintain forwarding tables.
- (iii) The scheme does not guarantee to deliver a copy of a broadcast packet to each destination.

(e) Forwarding along the MST

Advantages:

- (i) Simple to implement, if fixed. Requires little table space; just a list of branches incident to the node.

(ii) Minimizes total packet copies sent.

(iii) Implicitly avoids flooding and the need for duplicate detection.

(iv) Does not require multiple destination addressing.

Disadvantages:

(i) Not easily adapted to changing conditions.

(ii) Does not necessarily minimize propagation delay.

We have demonstrated the need for global and subgroup broadcast routing algorithms. The multi-destination addressing scheme is the most versatile, since it can be used satisfactorily in both environments, though the length of the header may become very long in large networks. There is some cutoff point after which it is more economical to use algorithms explicitly designed for global broadcast routing.

We feel that networks should provide a subgroup broadcast routing capability, in the form of multi-destination addressing, and a global broadcast routing capability in the form of MST forwarding or reverse path forwarding.

Reliable broadcast protocols must be built around these routing algorithms. Since not all applications require reliable broadcast, it may be appropriate for the hosts to provide this function. This is a subject for future research.

CHAPTER 4

PERFORMANCE EVALUATION OF BROADCAST ROUTING ALGORITHMS

4.1 Introduction

In this chapter we develop performance measures for broadcast routing in store-and-forward computer communication networks. These measures are used to evaluate the efficiency of the various broadcast routing algorithms described in the previous chapter.

The measures of performance are a function of the topology of the network, and can easily be determined for the various routing schemes for a given network. We would, however, like to have theoretical bounds on these measures for various classes of networks. The various classes of networks are distinguished by certain topological properties of the graphs that represent them, like the degree of the nodes, or whether the graph is regular or not [Harary69]. An alternate approach is to determine lower bounds for these measures for random graphs [Erdős73].

In this thesis we concentrate on determining lower bounds for networks that can be represented as regular graphs. Cerf, Cowan, Mullin and Stanton [Cerf74c] have shown that computer communication networks represented as regular graphs have useful properties regarding vulnerability, and shortest path lengths. They also show that by analyzing certain types of regular graphs, lower bounds on these properties can be derived. These lower bounds determine the shortest

path length and vulnerability for "ideal" networks against which network designers may measure their networks. For this reason, we too, determine the lower bounds on these performance measures for regular graphs. In addition, the analysis provides a uniform basis by which to compare the various algorithms.

Section 4.2 proposes a set of performance measures, and section 4.3 reviews and extends some properties of regular graphs. Sections 4.4 to 4.8 determine the performance of the various broadcast routing algorithms. Section 4.9 compares the various algorithms, and 4.10 determines average values of the performance measures for the ARPANET topology.

4.2 Performance Measures

This section develops some performance measures for broadcast communication in store-and-forward networks. We develop the performance measures by considering the overhead imposed on the communication subnet by a broadcast, and the delays in performing the broadcast.

An important measure of the amount of bandwidth used in performing a broadcast, and of the processing overhead in the switching (store-and-forward) nodes of the subnet is the Number of Packets Transmitted, $NPT(i)$, in broadcasting a message initiated from node i in the subnet. $NPT(i)$ is in units of packet-hops per broadcast.

The Broadcast Delay, $BD(i,j)$, is the delay before node j receives a broadcast message initiated from i . $BD(i,j)$ is in appropriate units for delay, e.g. seconds or hops. The average Broadcast Delay, $BDav(i)$, from a node i that initiates the broadcast is an estimate of the delay before a receiver hears the message. $BDav(i)$ is therefore a measure of the ability of the broadcast routing scheme to deliver messages quickly when initiated from i . In a communication subnet with N nodes

$$BDav(i) = \frac{1}{N-1} * \sum_{\substack{j=1 \\ j \neq i}}^N BD(i,j). \quad (4.1)$$

Analogously, $BDmax(i)$ is the maximum Broadcast Delay from a broadcaster i , and therefore is an indication of the maximum delay before all receivers have heard the message. $BDmax(i)$ is, therefore, also a measure of the cost of broadcast from node i .

$$BD_{\max}(i) = \max \{BD(1,j) \mid \forall 1 \leq j \leq N, j \neq i\}. \quad (4.2)$$

These measures are useful in the design of timeouts for the reinitiation of the broadcast, or recovery techniques in distributed operating systems that use the broadcast routing feature of the communication subnet to locate resources.

The Broadcast Cost, BC, in a communication network is the sum of the cost of broadcast with each node as the initiator. It is assumed that each node has equal probability of initiating a broadcast. $BD_{\max}(i)$ is the cost of broadcast when initiated from node i . Hence, if there are N nodes in the network, then

$$BC = \sum_{i=1}^N BD_{\max}(i). \quad (4.3)$$

These measures of performance can easily be determined for a particular network using a particular broadcast routing technique. The performance measures are a function of the topology of the network. In order to determine lower bounds on these measures, we examine regular graphs.

4.3 Regular Graphs

In this section we review and extend some of the definitions and properties of regular graphs, Moore graphs, and generalized Moore graphs.

We assume that the number of nodes in the graph is N , and that all edge costs are the same and equal to one. The degree of a node in a graph is the number of edges incident to that node. If the degree of all the nodes is same and equal to D , then the graph is called regular with degree D . Figure 4.1 shows some regular graphs. A theorem and two corollaries may be stated concerning the relationship between N and D in regular graphs. These are simple, and stated without proof.

Theorem 4.1: The sum of the degrees of the nodes of any graph is twice the number of edges in the graph.

Corollary 4.1.1: In any graph the number of nodes of odd degree is even.

Corollary 4.1.2: A regular graph with degree D and N nodes can always be constructed if $N \cdot D$ is even.

Consider a regular graph with degree D . The maximum number of nodes at distance one from any node is D , at distance two $D(D-1)$, at distance three $D(D-1)^2$ and so on. Moore graphs are those regular graphs that have exactly $D(D-1)^{j-1}$ nodes at distance j , for any node considered. Figure 4.2a illustrates such a tree as seen from one node X , where the graph has 10 nodes and is of degree 3. Such a picture can

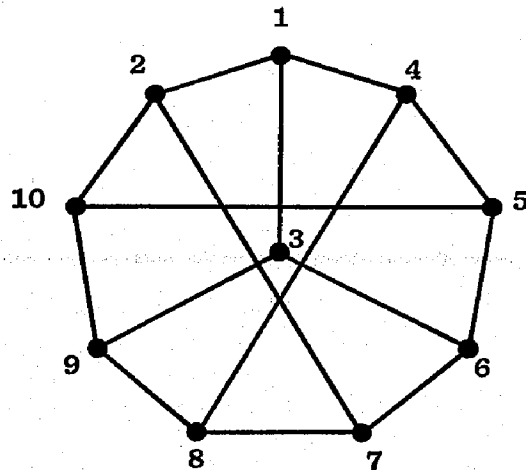
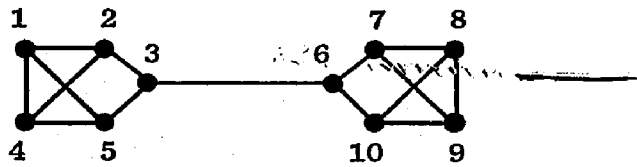
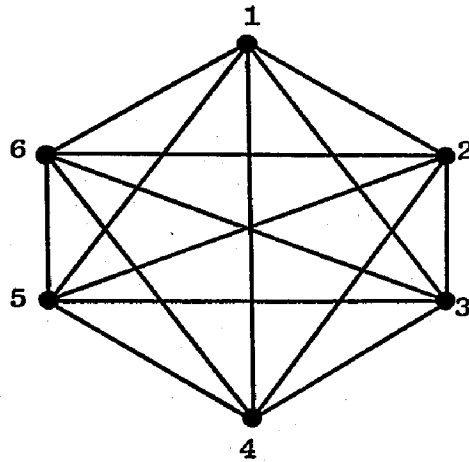


Figure 4.1. SOME REGULAR GRAPHS.

be drawn for all nodes in the graph. Thus one must join all nodes at the highest numbered level together in such a way to meet this constraint. This creates a sort of "spaghetti junction" at the highest level. Figure 4.2b illustrates how the tree of figure 4.2a can be converted into a Moore graph. Figure 4.2c is a more aesthetic representation of figure 4.2b. This is the Petersen Graph. Moore graphs satisfy the following relation:

$$N = 1 + D * \sum_{j=1}^m (D-1)^{j-1}. \quad (4.4)$$

where j is the level number and m the highest level. Conditions under which a given graph can be a Moore graph have been derived in [Hoffman60, Cerf73].

Of course, not all graphs have a number of nodes that satisfy equation (4.4). In this case the extra nodes are placed at the highest numbered level, m , and one must then join the nodes at level m and $m-1$ such that the constraints are satisfied. Such graphs are called generalized Moore graphs and have been studied in detail by Cerf, Cowan, Mullin and Stanton [Cerf73, Cerf74d, Cerf74e]. A generalized Moore graph having 16 nodes and of degree 3 is illustrated in figure 4.3.

We now introduce a new term. A pseudo generalized Moore graph is a regular graph such that for at least one node, the number of nodes at distance j is $D(D-1)^{j-1}$ for all $j > 1$ except possibly the highest numbered level which may be incompletely filled. In other words, from the point of view of that node, the graph looks like a filled tree except for the

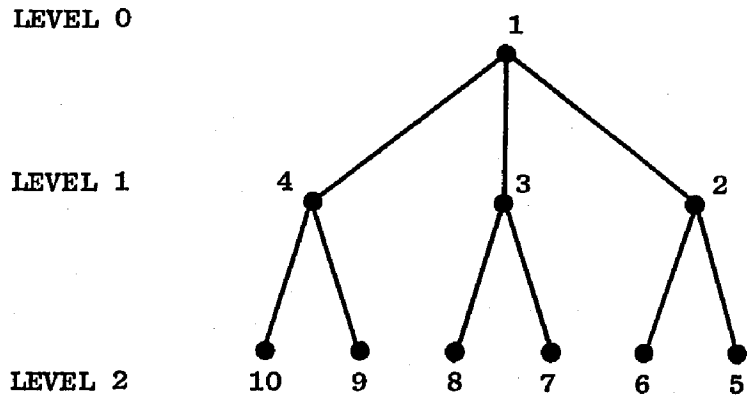


Figure 4.2a

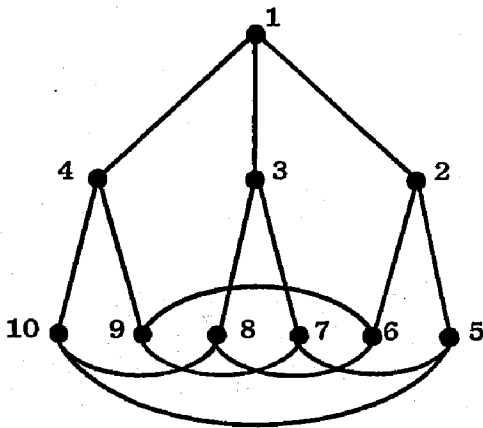


Figure 4.2b

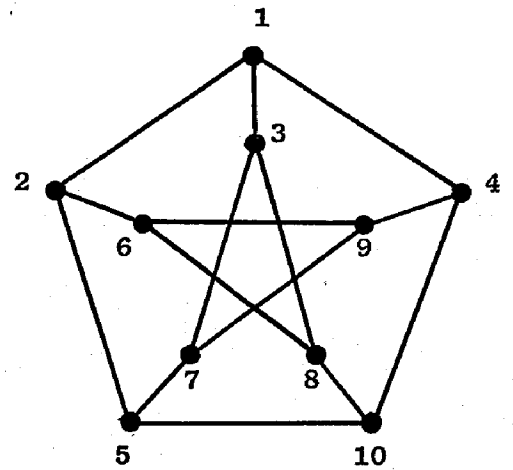


Figure 4.2c

Figure 4.2. A MOORE GRAPH.

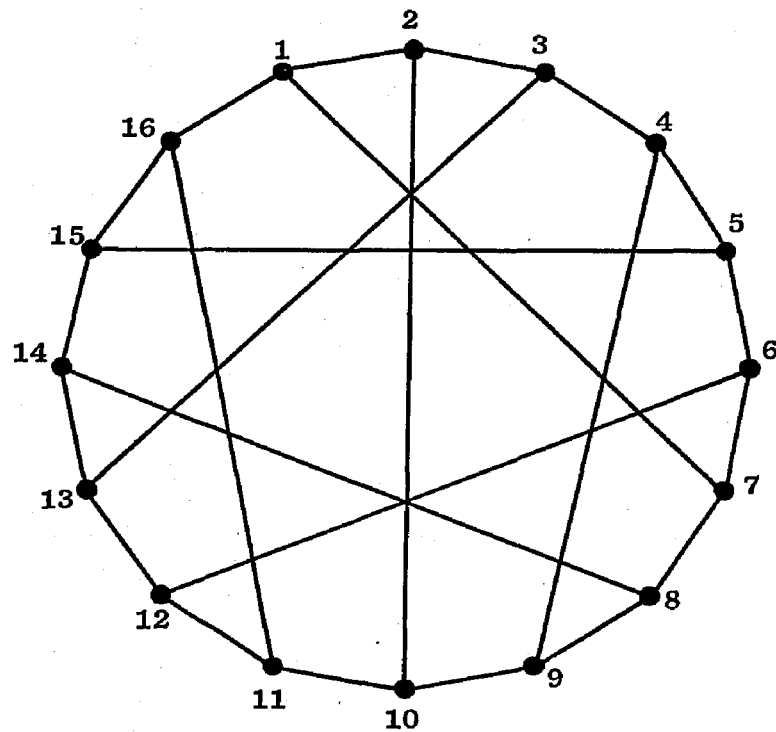


Figure 4.3. A GENERALIZED MOORE GRAPH HAVING 16 NODES AND DEGREE 3.

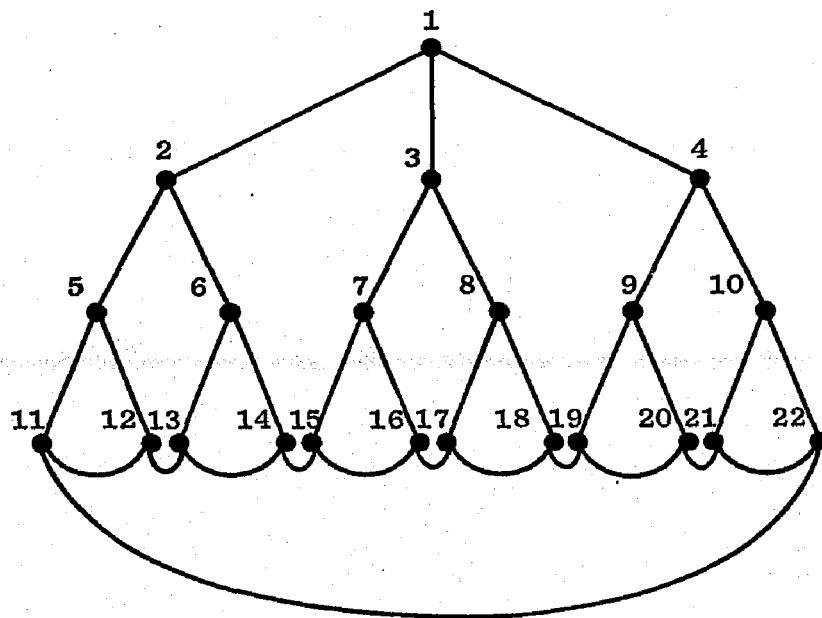


Figure 4.4. A PSEUDO GENERALIZED MOORE GRAPH HAVING 22 NODES AND OF DEGREE 3.

spaghetti junction at the last two levels. Figure 4.4 illustrates a pseudo generalized Moore graph of degree 3 having 22 nodes. A Moore graph does not exist for such a regular graph [Cerf73], even though its highest numbered level is completely filled, and so neither does a generalized Moore graph.

Theorem 4.2: For a given N and D such that a regular graph exists, a pseudo generalized Moore graph also exists.

Proof: The theorem is proved by showing that a pseudo generalized Moore graph can always be constructed if a regular graph can.

Take any node and connect it to D other nodes. Take each of these D nodes and connect each to $D-1$ other nodes that have not already been connected to, so as to form a tree. Continue connecting the leaves of the tree to $D-1$ other nodes until all N nodes have been used. The result is a minimum height tree with the starting node as the root. The highest level m may be incompletely filled, where $m \geq 1$.

We now show that if $N \cdot D$ is even then the nodes at level m and $m-1$ that do not have degree D can be connected to one another so as to produce a regular graph.

Let U be the number of nodes by which the m^{th} level is unfilled. If the m^{th} level was completely filled then it would have $D(D-1)^{m-1}$ nodes that would give rise to $D(D-1)^m$ branches. However, there are U less nodes at the m^{th} level and therefore the m^{th} level gives rise to $U(D-1)$ less branches, but the $(m-1)^{\text{th}}$ level

gives rise to an additional U . Therefore the total number of branches that must be connected to one another

$$\begin{aligned} &= D(D-1)^m + U - U(D-1) \\ &= D(D-1)^m - U(D-2). \end{aligned}$$

If this number is always even, then the branches can be appropriately paired off producing a regular graph. The first term, $D(D-1)^m$ is always even for any D since $m \geq 1$. When D is even, the second term is also even. We now show that when D is odd, U is even causing the second term also to be even, thereby proving the theorem. We now determine U . It is equal to the total number of nodes if the m^{th} level was full minus the actual number of nodes.

$$\begin{aligned} U &= \sum_{j=1}^m D(D-1)^{j-1} - N + 1 \\ &= \frac{D[(D-1)^m - 1]}{D-2} - N + 1 \\ &= \frac{D(D-1)^m - D + D - 2}{D-2} - N \\ &= \frac{D(D-1)^m - 2}{D-2} - N. \end{aligned}$$

Since D is odd, N must be even (Corollary 4.1.2). $D(D-1)^m$ is even for all $m \geq 1$. Since $D-2$ is odd and U is an integer, the first term is even. Therefore U is even.

Q.E.D.

From the definitions just introduced we see that Moore graphs are generalized Moore graphs, which in turn are pseudo generalized Moore graphs which themselves are regular graphs. We now determine lower bounds for the diameter and average path length in regular graphs.

4.3.1 Some Lower Bounds in Regular Graphs

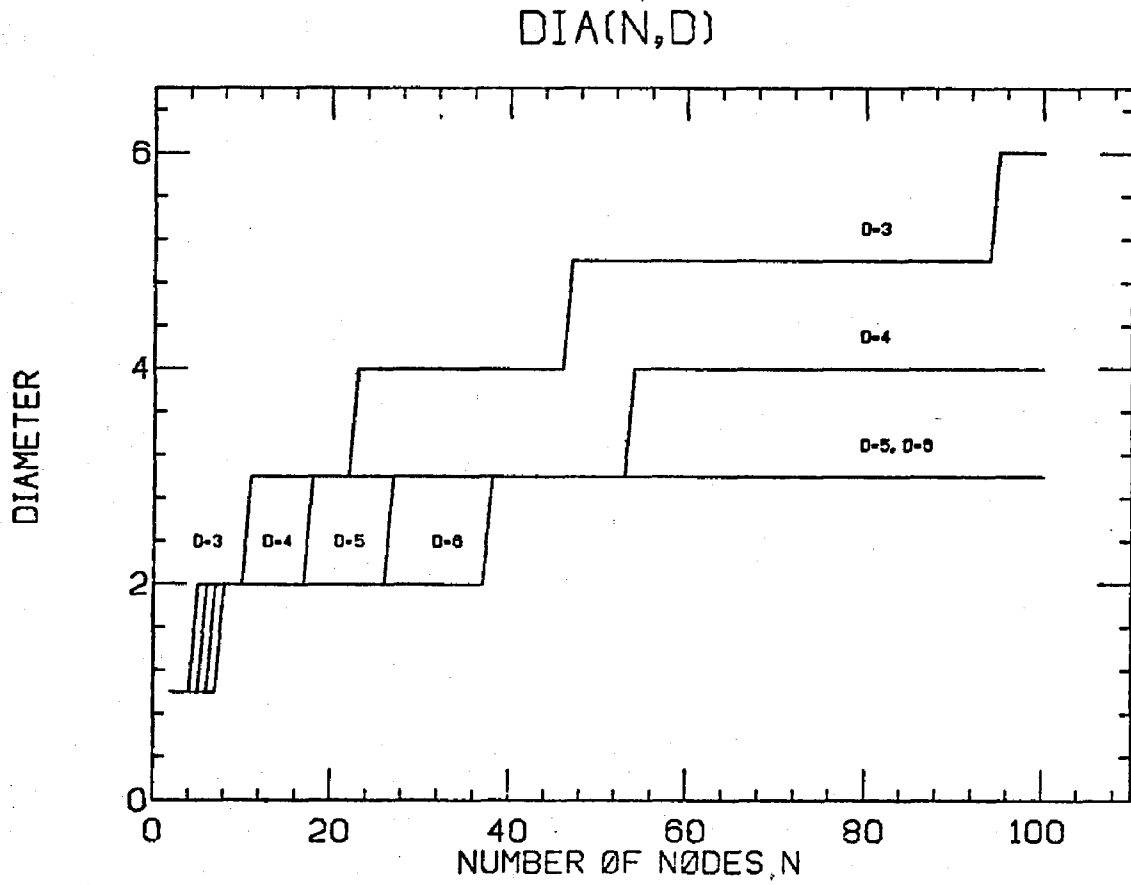
Cerf, et al, [Cerf74b, Cerf74c] have derived expressions for the lower bound on the diameter and average path length in regular graphs. They do so by analyzing the tree of a pseudo generalized Moore graph of degree D with N nodes. Note that for some D and N , regular graphs may not exist, but a minimum height tree does and so the analysis is valid.

The lower bound on the diameter of regular graphs, $DIA(N,D)$ is determined by finding the height of a minimum height tree connecting the N nodes of degree D .

$$DIA(N,D) = m = \left\lceil \log_{D-1} \frac{N*(D-2) + 2}{D} \right\rceil \text{ for } D > 2. \quad (4.5)$$

A plot of $DIA(N,D)$ is shown in figure 4.5.

The sum of the shortest path lengths from the root of the tree to all the other nodes can easily be determined. We first assume that the m^{th} level is completely filled and then subtract the contribution by the nodes that are absent. The lower bound on the sum of the shortest path lengths for a regular graph, SPL_{sum} , is equal to the sum of the shortest path lengths from the root. Therefore



$$\text{SPLsum}(N,D) = D * \sum_{j=1}^m j * (D-1)^{j-1} - (1 + D * \sum_{j=1}^m (D-1)^{j-1} - N) * m. \quad (4.6)$$

The dependence of SPLsum on N, and D is illustrated in figure 4.6.

The lower bound on the average shortest path length in regular graphs, SPLav, is analogously equal to the average shortest path length from the root of the tree, and is:

$$\text{SPLav}(N,D) = \frac{\text{SPLsum}(N,D)}{N-1}. \quad (4.7)$$

Figure 4.7 illustrates this relation.

These bounds are met by generalized Moore graphs, since for every node, the shortest path lengths look like the tree just described. In general these bounds will not be met by all regular graphs.

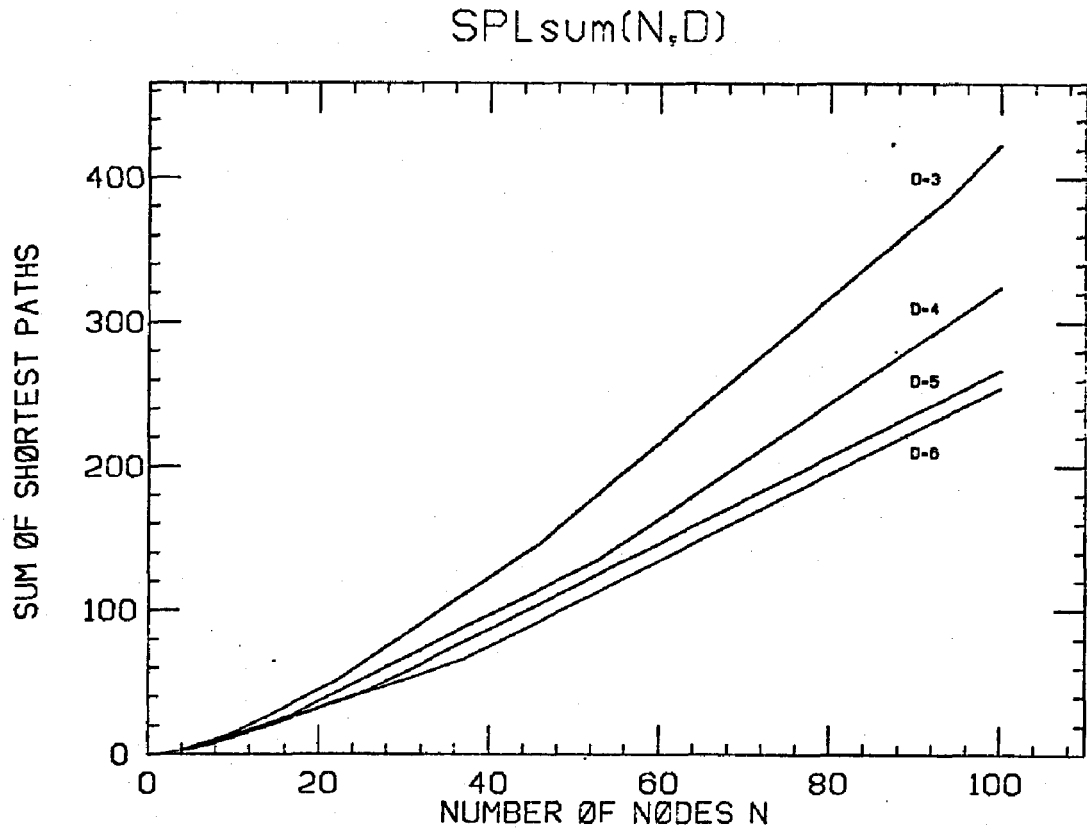


Figure 4.6. LOWER BOUND ON THE SUM OF SHORTEST PATH LENGTHS IN REGULAR GRAPHS.

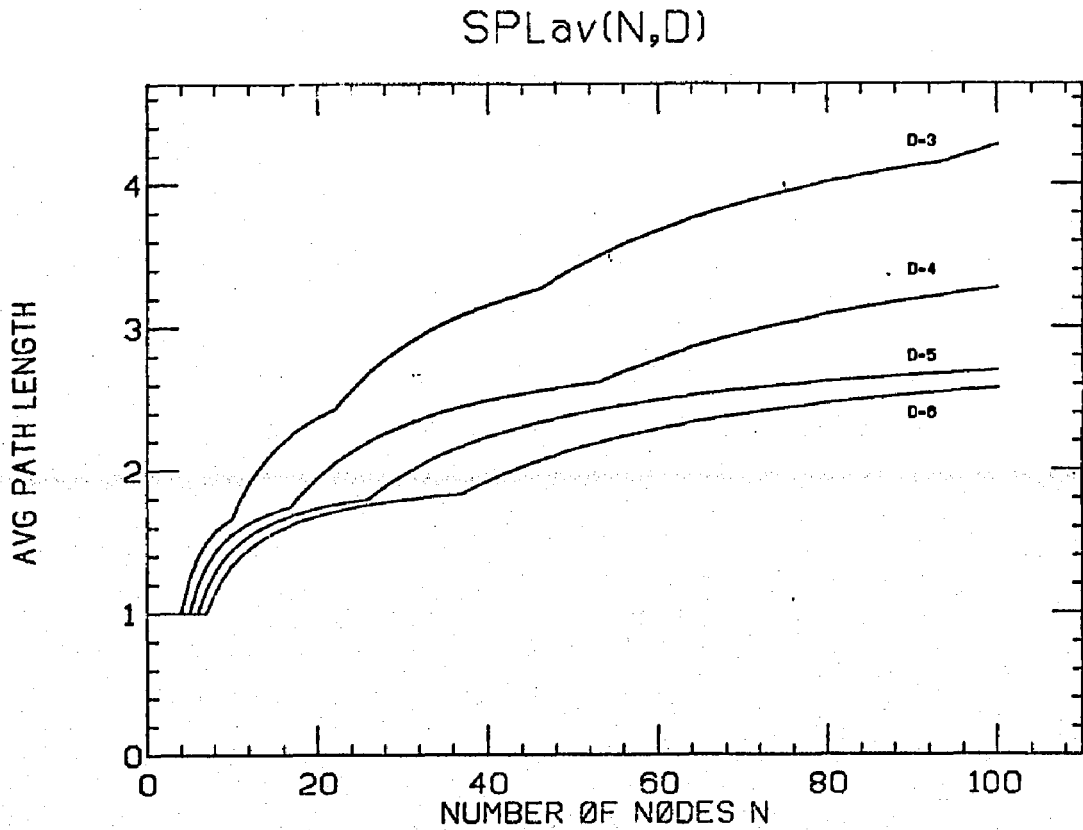


Figure 4.7. LOWER BOUND ON THE AVERAGE PATH LENGTH IN REGULAR GRAPHS.

4.4 Separately Addressed Packets (SAP)

In this section we determine lower bounds* for NPT, BDav, BDmax and BC for regular graphs when the broadcast routing strategy is to transmit separately addressed packets from the source to each destination. We assume that the routing of the packets is along the shortest path (in the graph theoretical sense) from the broadcaster to the destinations.

4.4.1 Number of Packets Transmitted

Assume that each broadcast message consists of a single packet. Hence the number of packet-hops from a broadcaster to a receiver is equal to the path length from the broadcaster to the receiver. Therefore a lower bound on the number of packets transmitted to achieve the broadcast is equal to the lower bound on the sum of the path lengths. Therefore

$$NPT(N,D)_{SAP} = SPL_{sum}(N,D). \quad (4.8)$$

The dependence of NPT_{SAP} on N and D is illustrated in figure 4.6.

4.4.2 Broadcast Delay

In order to determine the lower bound on the various broadcast delays when transmitting separately addressed packets, we must determine the order in which messages (packets) are transmitted from the

*Since lower bounds are being determined, i, will be dropped from $NPT(i)$, $BDav(i)$ and $BDmax(i)$.

broadcaster to the various receivers so that the maximum broadcast delay is minimized. Since the lower bounds are being determined by considering the root node of a pseudo generalized Moore graph, the optimal paths lie along the branches of the tree connecting the root node to the various other nodes. Figure 4.8a illustrates such a minimum height tree for a node X. The subtrees of X form the primary subtrees of this tree. We assume that X can be transmitting packets to all primary subtrees at once, and hence we must consider the delays for a particular primary subtree - the longest one. Also assume that no other traffic is present to interfere with the broadcast packets.

The rate at which packets may be transmitted to nodes in a primary subtree is determined by the transmission delays of the communication link. Let us assume that this is a constant and that all times are relative to this quantity. In order to minimize the time taken to broadcast the packets to all nodes of the primary subtree, the root X must transmit packets to the node farthest away first and so on. This ensures that there will be as many packets as possible in transit at the same time, thus minimizing the total broadcast delay. If node X started transmitting a packet at time zero, and then one more at every time interval, then the last packet would be transmitted at time equal to the number of nodes in the primary subtree minus one. This packet would arrive at its destination in one time unit, since its destination is the closest node - the root of the primary subtree. Hence the time taken for all nodes in the primary subtree to get the broadcast message is equal to the number of nodes in the primary subtree. Figure 4.8b

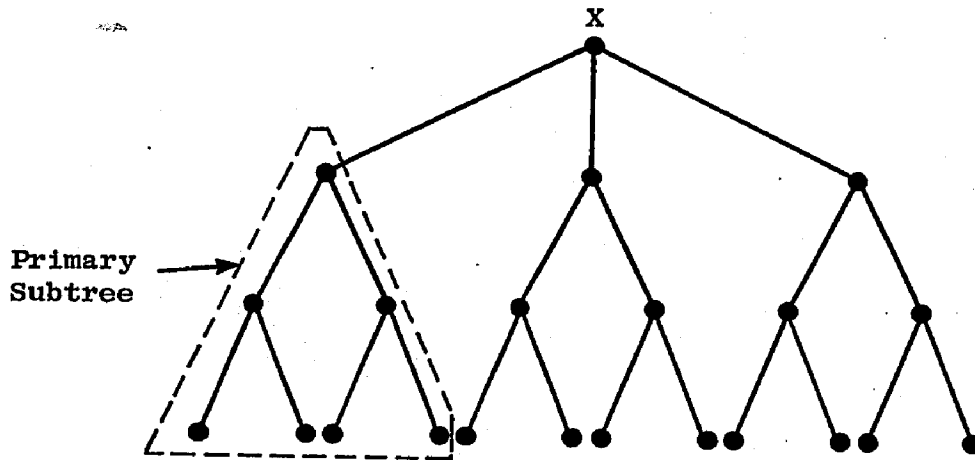


Figure 4.8a. SHORTEST PATH TREE FOR ROUTING.

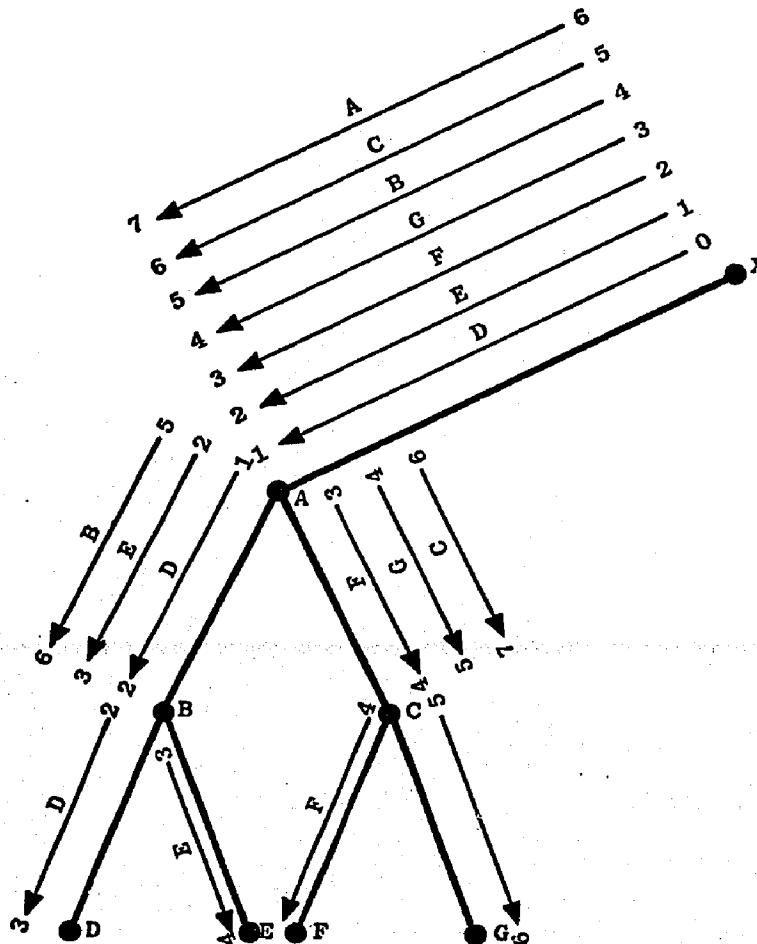


Figure 4.8b. ORDER IN WHICH PACKETS ARE TRANSMITTED.

illustrates the order in which the packets are transmitted for the largest primary subtree of the tree in figure 4.8a.

We now proceed to find the number of nodes in the largest primary subtree of a pseudo generalized Moore graph.

Let R be the number of nodes remaining in the unfilled level, should there be one, and let h be the highest numbered completely filled level.

$$\begin{aligned} h &= m-1 && \text{if } U \neq 0 \\ &= m && \text{if } U = 0 \end{aligned}$$

and
$$R = N-1 - \sum_{j=1}^h D(D-1)^{j-1}.$$

If R is equal to zero, then the number of nodes in each primary subtree is the same and equal to

$$\sum_{j=1}^m (D-1)^{j-1}.$$

However, R may not be equal to zero, and since we want to find the lower bound on delay, we want to determine the minimum number of nodes in the largest primary subtree. Hence the R remaining nodes (and hence, the U unfilled places) must be distributed as equally as possible among the primary subtrees. The minimum number of unfilled places any primary subtree can have is $\lfloor U/D \rfloor$. Therefore the minimum number of nodes in the largest primary subtree, and hence $BD_{\text{max}}^{\text{SAP}}$ is

$$BD_{\max}(N,D)_{SAP} = \sum_{j=1}^m (D-1)^{j-1} - \lfloor U/D \rfloor \quad (4.9)$$

and

$$BC(N,D)_{SAP} = N * BD_{\max}(N,D)_{SAP}. \quad (4.10)$$

These relationships are illustrated in figures 4.9, and 4.10 respectively.

In order to find the average broadcast delay we must find the delay to all nodes and then divide by the number of receivers. For the moment assume that all primary subtrees are completely filled, (later we will remove this assumption). It was shown that packets were transmitted from the root at times $0, 1, 2, \dots, (\sum_{j=1}^m (D-1)^{j-1} - 1)$ if the highest level is completely filled. These packets are first destined for nodes in level m , then $m-1, \dots$ and finally level 1. Hence the time taken for a packet to reach its destination is given by the time at which it was transmitted plus the level number in which the destination lies. Let NPS be the number of nodes in a completely filled primary subtree.

$$NPS = \sum_{j=1}^m (D-1)^{j-1}.$$

The sum of delays for all D completely filled primary subtrees is:

$$D * \sum_{j=1}^m \sum_{i=LB}^{UB} (j + NPS - i).$$

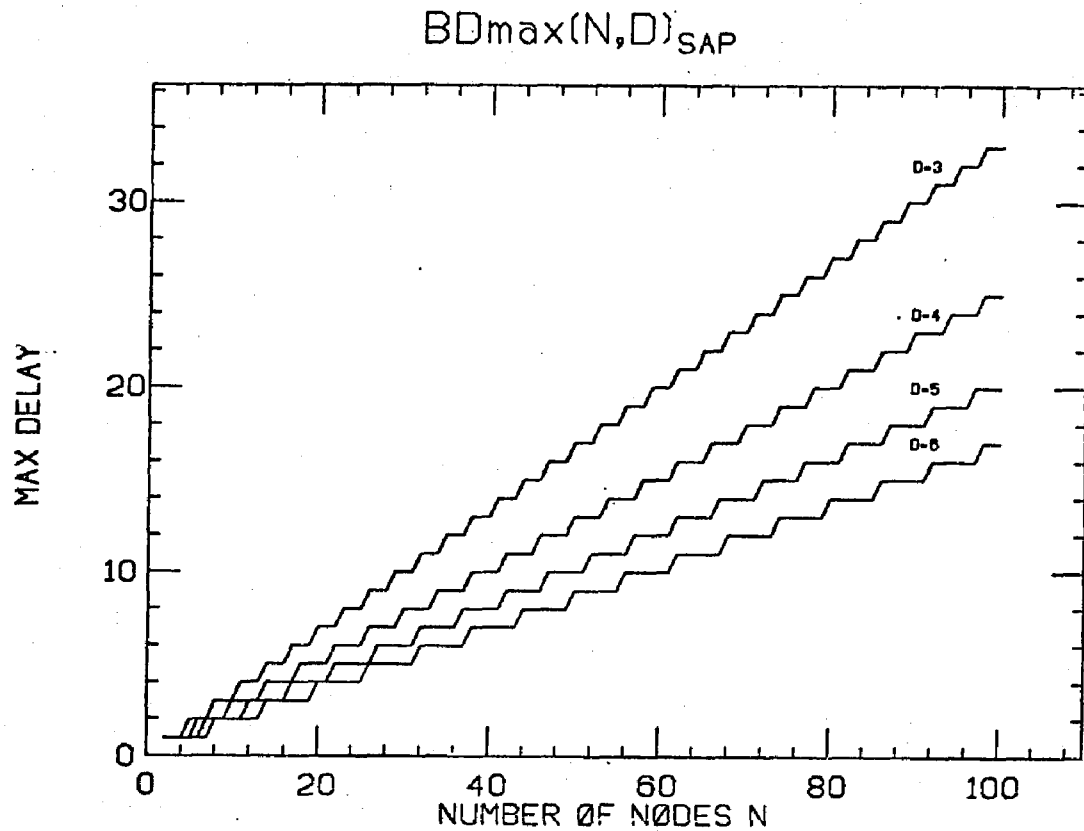


Figure 4.9. LOWER BOUND ON MAXIMUM BROADCAST DELAY FOR SEPARATELY ADDRESSED PACKETS.

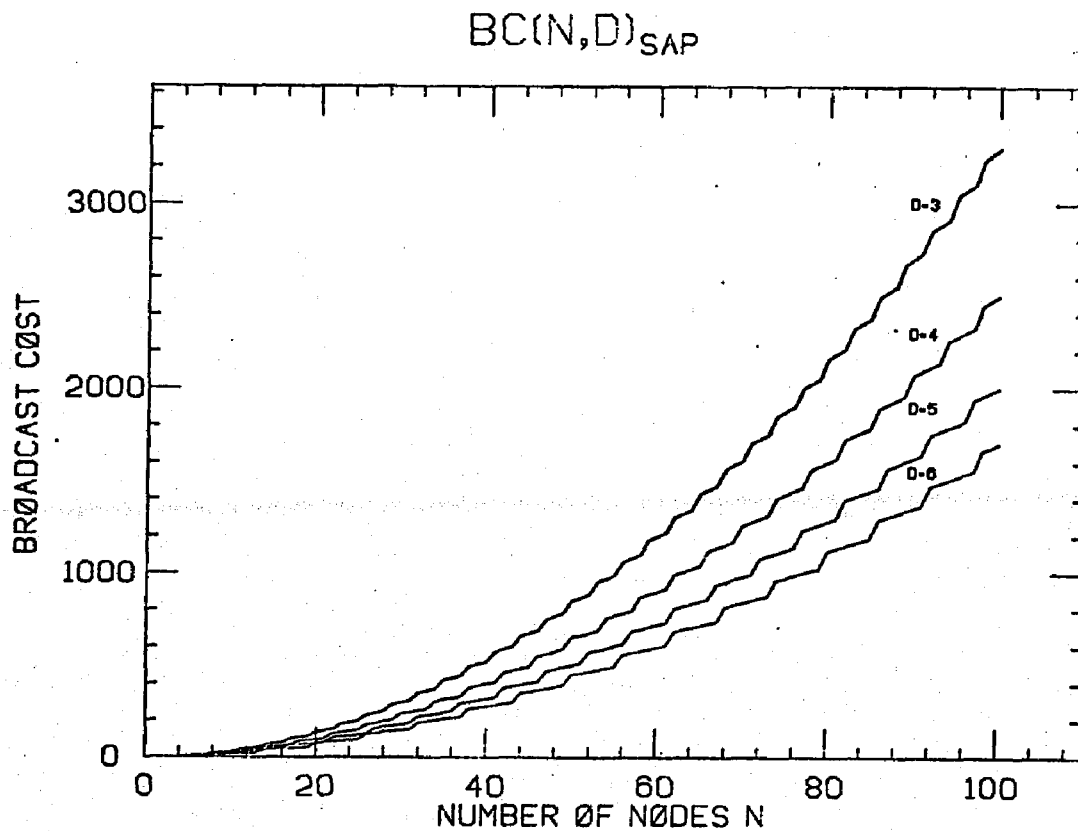


Figure 4.10. LOWER BOUND ON BROADCAST COST FOR SEPARATELY ADDRESSED PACKETS.

where

$$LB = 1 + \sum_{k=1}^{j-1} (D-1)^{k-1}$$

and

$$UB = \sum_{k=1}^j (D-1)^{k-1}.$$

The outer summation cycles through each of the levels, while the inner summation cycles through the nodes within a level. The limits of the inner summation are themselves summations dependent on j since the number of nodes in a level depends on the level number. The simplification of the summation can be found in Appendix D, and we just state the result here:

$$D \cdot NPS \cdot \frac{(NPS-1+2m)}{2} - \frac{D \cdot (NPS-m)}{D-2}.$$

In general, however, U will not be equal to zero, and so the reduction in delay owing to these U unfilled places must be determined. Each unfilled place causes the reduction of one unit of delay from all nodes in its primary subtree except those at its own level. This is readily apparent since the root is able to transmit all the packets in a shorter time. This reduction amounts to:

$$U \cdot \sum_{j=1}^{m-1} (D-1)^{j-1}.$$

The reduction in total delay owing to the U unfilled places themselves must now be determined. These unfilled positions were equally

distributed among the primary subtrees. Therefore we must cycle through the primary subtrees removing the node with the largest delay each time.

The largest delay a node in a primary subtree can have is:

$$m + (D-1)^{m-1} - 1.$$

Therefore the reduction in total delay is

$$\begin{aligned} & \sum_{i=0}^{U-1} (m + (D-1)^{m-1} - 1 - \lfloor i/D \rfloor) \\ &= U * \{ (m-1) + (D-1)^{m-1} \} - \sum_{i=0}^{U-1} \lfloor i/D \rfloor. \end{aligned}$$

The number of complete cycles through the D primary subtrees is $\lfloor U/D \rfloor$. During each one of these cycles, D nodes are removed; finally leaving $U - D * \lfloor U/D \rfloor$ nodes.

$$\begin{aligned} \therefore \sum_{i=0}^{U-1} \lfloor i/D \rfloor &= \sum_{j=1}^{\lfloor U/D \rfloor} D(j-1) + \lfloor U/D \rfloor * \{ U - D * \lfloor U/D \rfloor \} \\ &= (D/2) * \lfloor U/D \rfloor * \{ \lfloor U/D \rfloor - 1 \} + U * \lfloor U/D \rfloor - D * \lfloor U/D \rfloor^2 \\ &= \lfloor U/D \rfloor * \{ (D/2) * \lfloor U/D \rfloor - (D/2) + U - D * \lfloor U/D \rfloor \} \\ &= \lfloor U/D \rfloor * \{ U - (D/2) * \lfloor U/D \rfloor - (D/2) \}. \end{aligned}$$

$$\therefore \text{Total Delay} = D \cdot \text{NPS} \cdot (\text{NPS} - 1 + 2m) / 2 - D \cdot (\text{NPS} - m) / (D - 2)$$

$$- U \cdot \sum_{j=1}^{m-1} (D-1)^{j-1} - U \cdot \{ (m-1) + (D-1)^{m-1} \}$$

$$- \lfloor U/D \rfloor \cdot \{ U - (D/2) \cdot \lfloor U/D \rfloor - (D/2) \}.$$

$$\therefore \text{BDav}(N, D)_{\text{SAP}} = \text{Total Delay} / (N-1). \quad (4.11)$$

This relation is shown in figure 4.11.

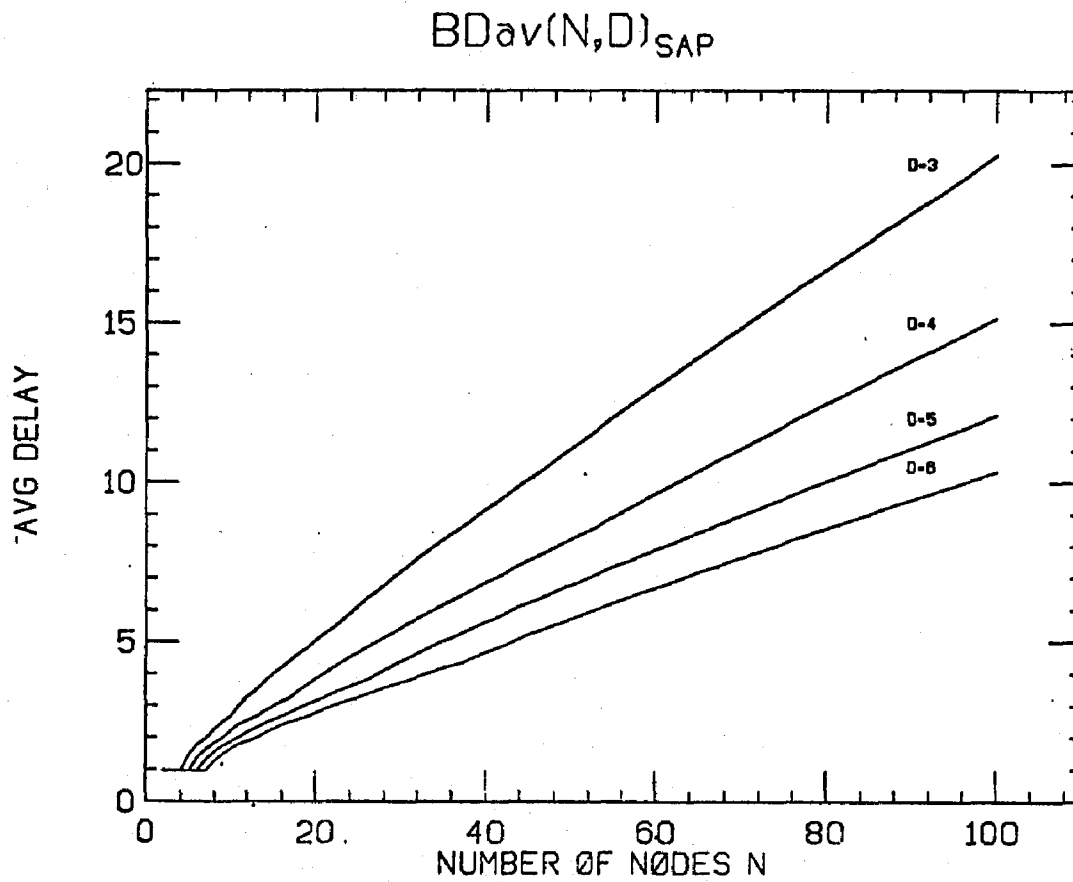


Figure 4.11. LOWER BOUND ON AVERAGE BROADCAST DELAY FOR SEPARATELY ADDRESSED PACKETS.

4.5 Multi-Destination Addressing (MDA) and Source Based Forwarding (SBF)

This section determines the performance measures when the broadcast routing technique is either multi-destination addressing or source based forwarding. The two techniques have the same performance since they both forward the packets along the optimal paths from the broadcaster to the receivers.

The number of packets transmitted is equal to the minimum number of edges by which the broadcaster is connected to the all the destinations. This is $N-1$. Therefore

$$NPT(N,D)_{MDA} = NPT(N,D)_{SBF} = N-1. \quad (4.12)$$

From the view point of the node initiating the broadcast the rest of the network looks like a minimum height tree. This tree is similar to the tree of a pseudo generalized Moore graph. Therefore we have:

$$BDav(N,D)_{MDA} = BDav(N,D)_{SBF} = SPLav(N,D) \quad (4.13)$$

$$BDmax(N,D)_{MDA} = BDmax(N,D)_{SBF} = DIA(N,D) \quad (4.14)$$

$$BC(N,D)_{MDA} = BC(N,D)_{SBF} = N * BDmax(N,D)_{MDA}. \quad (4.15)$$

These relationships are illustrated in figures 4.7, 4.5, and 4.12 respectively.

We have not determined the number of packet copies transmitted and the various delays if a restricted multi-destination addressing scheme

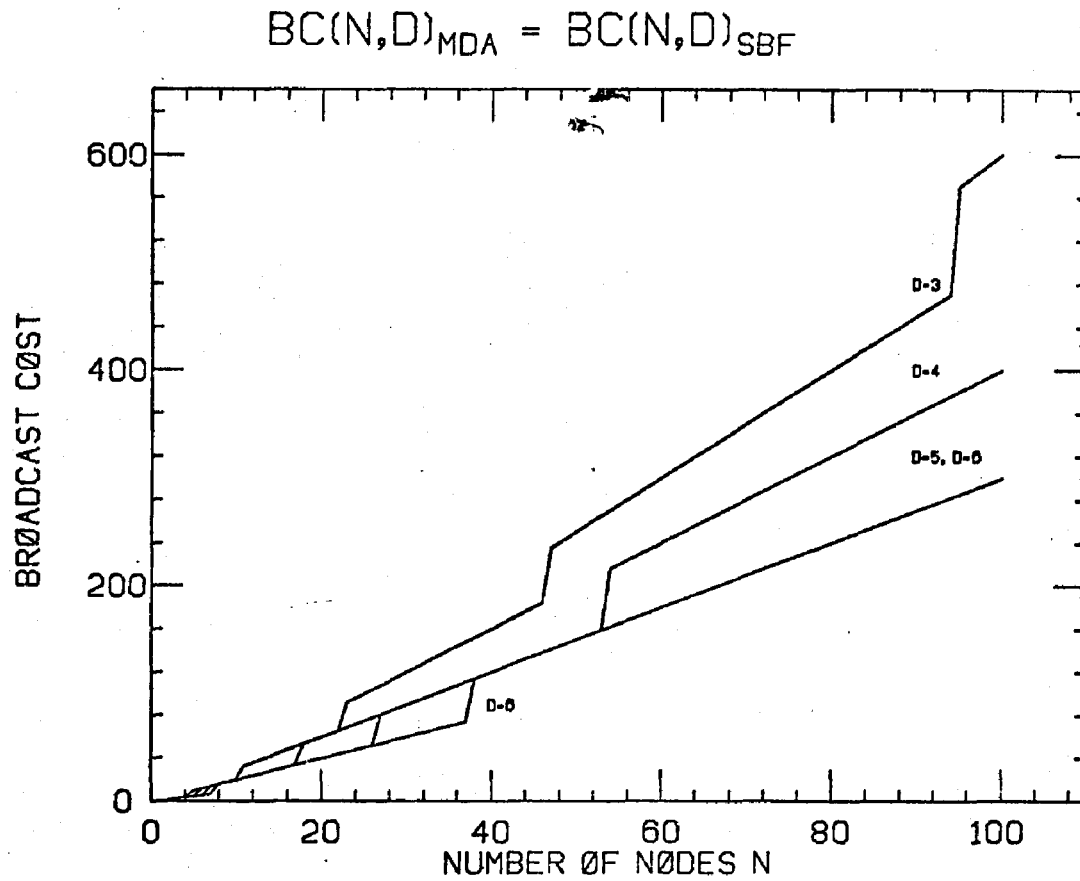


Figure 4.12. LOWER BOUND ON BROADCAST COST FOR MULTI-DESTINATION ADDRESSING AND SOURCE BASED FORWARDING.

were to be used. This is a subject for future research. The analysis parallels that used to determine the performance measures if separately addressed packets were transmitted (cf. section 4.4). One must determine the optimal order in which destinations are assigned, to the restricted multi-destination address field in the header, and the optimal order in which the packets are transmitted. This formulation would then include the analysis for separately addressed packets and multi-destination addressing as special cases, where the length of address field is one, or as long as desired, respectively.

4.6 Hot Potato Forwarding (HPF)

We analyze the hot potato broadcast routing scheme in which each packet maintains a hop count, and if this ever exceeds the discard threshold, DT , then the packet is discarded in order to prevent the subnet from flooding.

Since there is a discard threshold after which packets will be discarded, every packet transmitted from the broadcaster along the various links will be forwarded DT times. From the source, the packet will be transmitted D times, and then each node that receives a packet will forward it $(D-1)$ times. Therefore, the total number of packets transmitted is:

$$NPT(N,D)_{HPF} = \sum_{j=1}^{DT} D*(D-1)^{j-1}.$$

This is the actual number of packet copies transmitted and not a bound. If the discard threshold was $N-1$, then the upper bound on NPT would be:

$$NPT(N,D)_{HPF}(\text{upper}) = D*\{(D-1)^{(N-1)} - 1\}/(D-2). \quad (4.16)$$

This is illustrated in figure 4.13. Notice how quickly the number of packet copies transmitted per broadcast becomes very large for even a small sized network.

The smallest safe value of DT is equal to the diameter of the graph. The lower bound on the diameter of regular graphs is given by equation (4.5). Therefore a lower bound on $NPT(N,D)$ is:

$$NPT(N,D)_{HPF}(\text{lower}) = D \cdot \{(D-1)^{DIA(N,D)} - 1\} / (D-2). \quad (4.17)$$

This is illustrated in figure 4.14.

Since a packet is being forwarded along all links except the one on which it arrived, $BDav$ and $BDmax$ are identical to those for the multi-destination routing, or the source based forwarding case.

Therefore

$$BDav(N,D)_{HPF} = SPLav(N,D) \quad (4.18)$$

$$BDmax(N,D)_{HPF} = DIA(N,D) \quad (4.19)$$

$$BC(N,D)_{HPF} = N \cdot DIA(N,D). \quad (4.20)$$

These dependencies are illustrated in figures 4.7, 4.5 and 4.12 respectively.

Note that in the determination of $BDav$, $BDmax$ and BC , queueing delays arising from the interference of the extra packets were not taken into account. In all the other broadcast routing schemes discussed so far, $BDmax$ was also a measure of the time during which packets of a particular broadcast message would remain in the network. This is so because extra packets were not generated by the network in an attempt to forward the packets to all the destinations. In the hot potato broadcast routing scheme $BDmax$ is not a measure of how long packets for a particular broadcast will remain in the subnet. It is difficult to determine analytically what this time is. Intuitively, however, an estimate can be determined if queueing delays are neglected. A

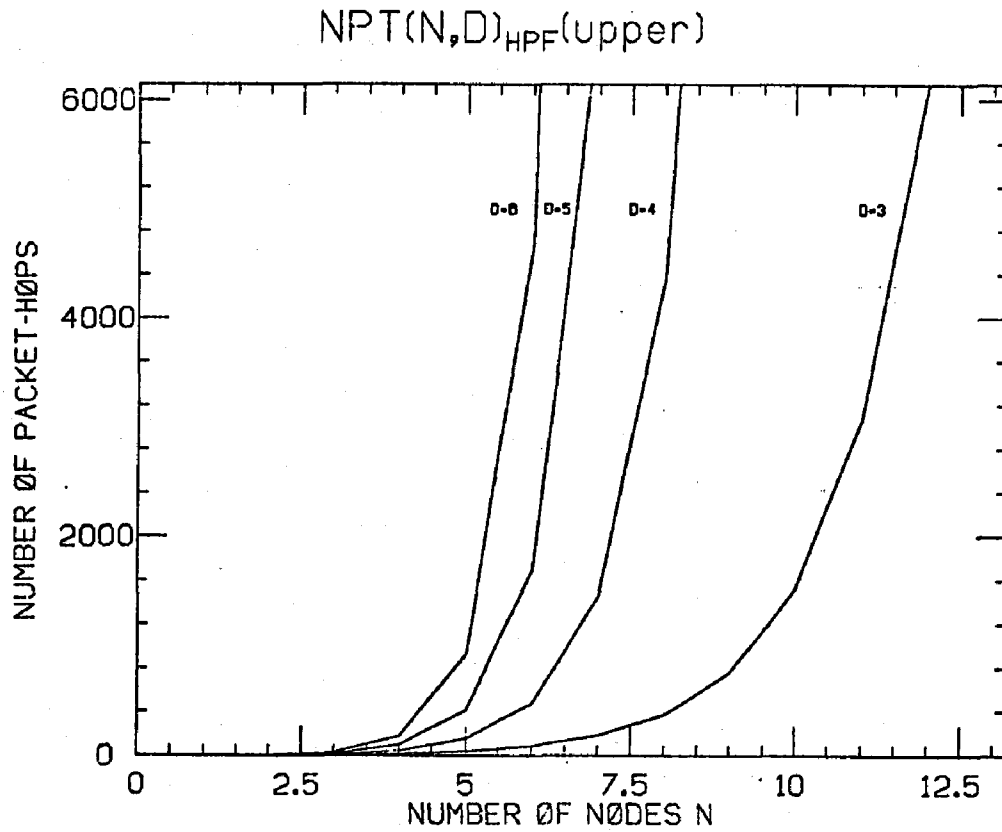


Figure 4.13. UPPER BOUND ON THE NUMBER OF PACKETS TRANSMITTED FOR HOT POTATO FORWARDING WITH DISCARD THRESHOLD.

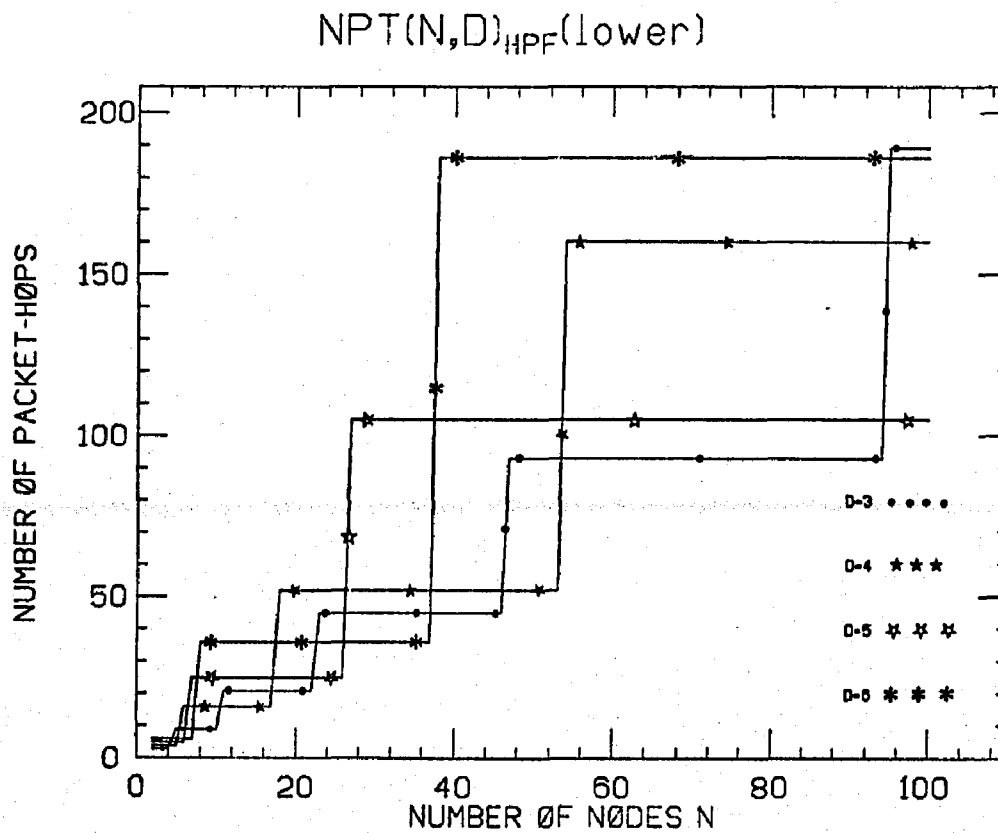


Figure 4.14. LOWER BOUND ON THE NUMBER OF PACKETS TRANSMITTED FOR HOT POTATO FORWARDING WITH DISCARD THRESHOLD.

particular packet transmitted from the broadcasting node will keep giving rise to newer packets DT times. Hence a very low lower bound on the time for which broadcast packets will remain in the subnet is DT .

4.7 Reverse Path Forwarding (RPF)

We have shown in section 3.6.2, that if the simple reverse path forwarding scheme is used, then each node except the source forwards exactly one of the arriving broadcast packet along all links except the one on which it arrived. The source transmits the broadcast packet along all links incident to it.

$$\therefore \text{NPT}(N,D)_{\text{RPF}}(\text{simple}) = N*(D-1) + 1. \quad (4.21)$$

This is illustrated in figure 4.15.

If the optimal version of the algorithm is used then broadcast packets are only forwarded along the branches of the reverse path tree and therefore

$$\text{NPT}(N,D)_{\text{RPF}}(\text{optimal}) = N-1. \quad (4.22)$$

In determining the lower bound on delays, since all edge costs are same and equal to unity the reverse path tree from a source is isomorphic to the shortest path tree. Therefore

$$\text{BDav}(N,D)_{\text{RPF}} = \text{SPLav}(N,D) \quad (4.23)$$

$$\text{BDmax}(N,D)_{\text{RPF}} = \text{DIA}(N,D) \quad (4.24)$$

$$\text{BC}(N,D)_{\text{RPF}} = N*\text{DIA}(N,D). \quad (4.25)$$

These dependencies are illustrated in figures 4.7, 4.5 and 4.12 respectively.

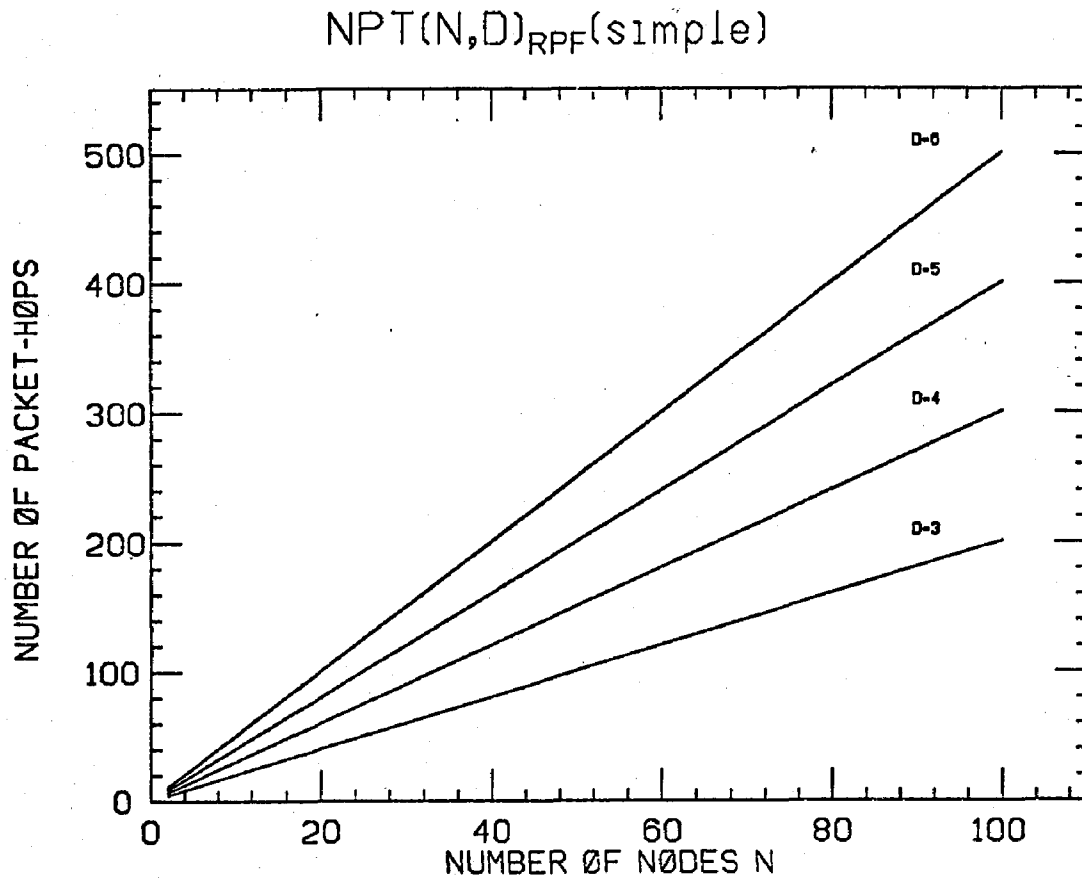


Figure 4.15. NUMBER OF PACKETS TRANSMITTED FOR SIMPLE REVERSE PATH FORWARDING.

Note that in the simple scheme, the delays are not dependent on the fact that extra packets are transmitted, since these packets are transmitted along links that are not part of the shortest path tree from the source. These packets, therefore, do not interfere with the packets that are forwarded along the branches of the tree. BD_{max} is, however, in this case not a measure of the time that a broadcast packet will remain in the network, but can be used as a rough approximation.

4.8 Minimal Spanning Tree Forwarding (MST)

We now determine the various performance measures when broadcast is performed along the branches of a minimum spanning tree that connects the various nodes of the network.

The number of packets transmitted is equal to the number of branches of the MST, which is $N-1$. Therefore

$$NPT(N,D)_{MST} = N-1. \quad (4.26)$$

4.8.1 Broadcast Delays

In order to determine the various performance measures based on delays, we must first determine what the MST in regular graphs looks like. The girth of a graph is the length of the smallest cycle or circuit that can occur in the graph. Cerf, Cowan, Mullin and Stanton have shown that the girth of a Moore graph is $2m+1$ [Cerf73], and of generalized Moore graphs is greater than or equal to $2m-1$ [Cerf74d]. The girth was determined by examining the minimum height tree and the permissible connections in the spaghetti junction. Hence, the tree obtained by removing one of the connections in the spaghetti junction results in a minimum diameter spanning tree that is also the minimum diameter MST. Since such a tree can always be constructed for pseudo generalized Moore graphs, the lower bound on the diameter of MSTs in regular graphs is $2m-2$. The performance measures are determined for these MSTs.

Lower bounds on B_{dav} and B_{dmax} are determined by finding $B_{\text{dav}}(X)$ and $B_{\text{dmax}}(X)$ where X is the broadcaster and is also the root of the tree. Therefore

$$B_{\text{dav}}(N,D)_{\text{MST}} = \text{SPLav}(N,D) \quad (4.27)$$

$$B_{\text{dmax}}(N,D)_{\text{MST}} = \text{DIA}(N,D). \quad (4.28)$$

These relationships are illustrated in figures 4.7 and 4.5 respectively.

The lower bound on the broadcast cost, BC , could have been calculated as:

$$BC(N,D)_{\text{MST}} = N * B_{\text{dmax}}(N,D)_{\text{MST}},$$

as has been done for the other broadcast routing techniques. However, for MSTs it is possible to find a much tighter lower bound by finding the sum of the maximum broadcast cost from every node. In order to perform this analysis it is necessary to make some assumptions on the position of the nodes in the highest unfilled level, should there be one. Since we are interested in finding a lower bound, the nodes of the unfilled level should be filled completely from either end as shown in figure 4.16.

Let PS be the number of primary subtrees containing nodes of the unfilled level. Since the R nodes are bunched up at either end, we have:

$$PS = \left\lceil \frac{R}{(D-1)h} \right\rceil$$

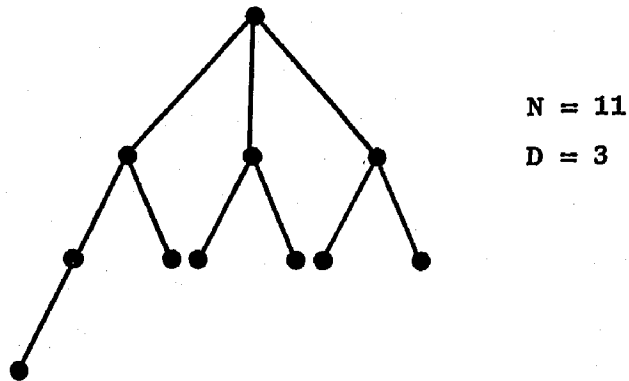
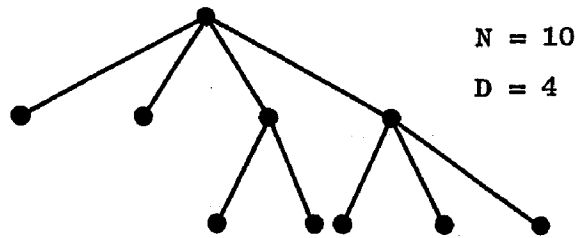
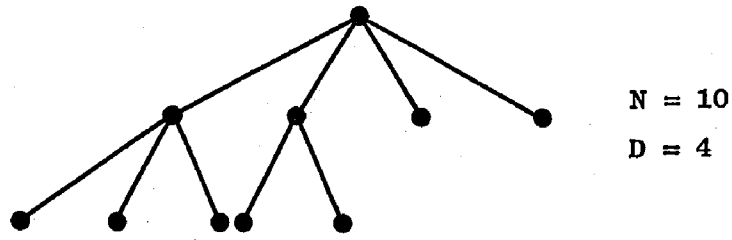


Figure 4.16. ARRANGEMENT OF NODES AT THE HIGHEST LEVEL.

where h is the highest completely filled level.

Let $\delta(x)$ be the unit impulse function defined as

$$\begin{aligned}\delta(x) &= 1 \quad \text{if } x=0 \\ &= 0 \quad \text{otherwise.}\end{aligned}$$

The maximum cost of broadcast from a node at level j , is equal to the level number, j , plus the height (highest level) of the longest primary subtree excluding the one containing the node. If PS is greater than one, then for all nodes the height of such a primary subtree is m .

The maximum cost of broadcast from the root is m . From each of the nodes in the unfilled level, the maximum cost of broadcast is $2m - \delta(PS-1)$. The maximum broadcast cost from all the other nodes is determined by first supposing that PS is greater than one, and then subtracting the extra delay if PS is equal to one. Therefore

$$\begin{aligned}BC(N,D)_{MST} &= m + R*(2m - \delta(PS-1)) \\ &+ \sum_{j=1}^h \{(m+j)*D*(D-1)^{j-1} - (D-1)^{j-1}*\delta(PS-1)\}.\end{aligned}\tag{4.29}$$

Figure 4.17 illustrates the dependence of BC_{MST} on N and D .

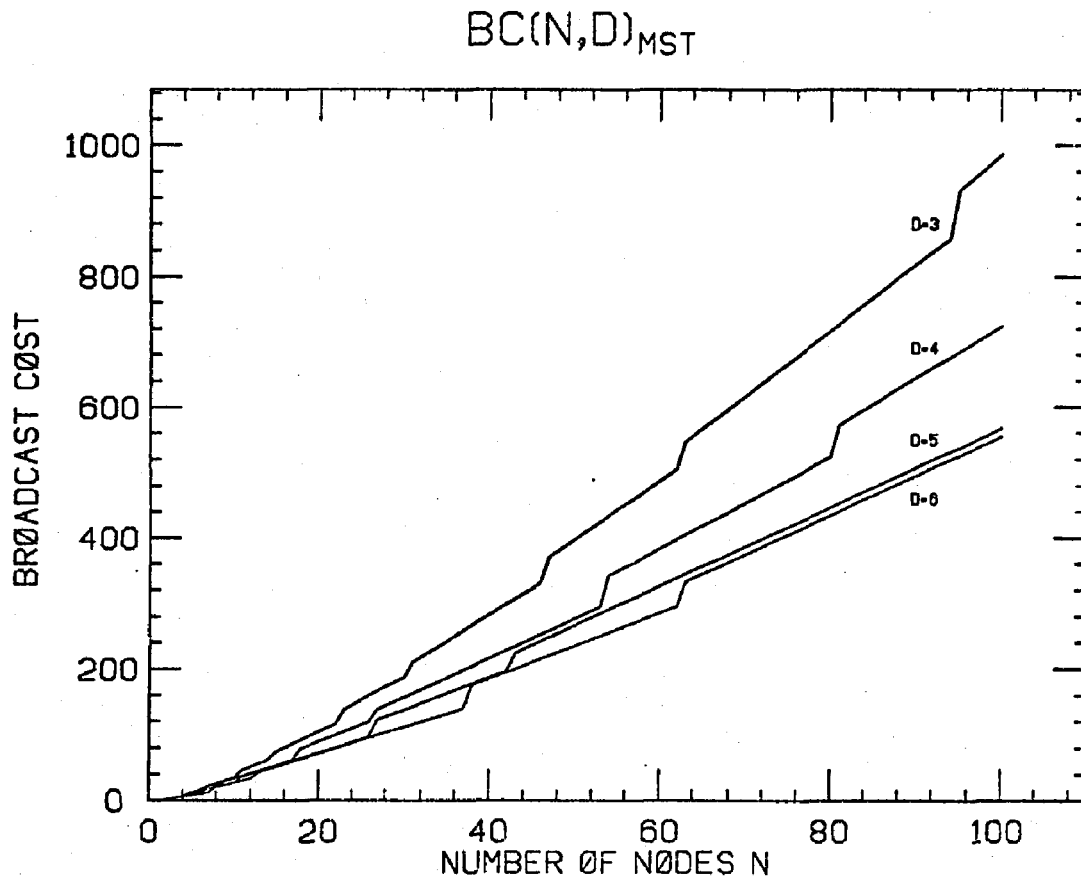


Figure 4.17. LOWER BOUND ON BROADCAST COST FOR MINIMAL SPANNING TREE FORWARDING.

4.9 Comparison of the Different Schemes

We now compare the six different broadcast routing algorithms. Lower bounds on NPT, BDav, BDmax and BC for the different algorithms, for a regular graph of degree 3 are shown in figures 4.18, 4.19, 4.20 and 4.21 respectively. These figures illustrate the behaviour of the performance measures for large N. We notice, that for a broadcast-to-all mode of communication, separately addressed packets is definitely not suitable owing to the large overhead on the communication subnet. Hot potato forwarding is, also not suitable because of the extra traffic it generates. Forwarding along the MST, multi-destination addressing, source based forwarding and reverse path forwarding are acceptable techniques. Their relative merits and demerits have been discussed in detail in the previous chapter.

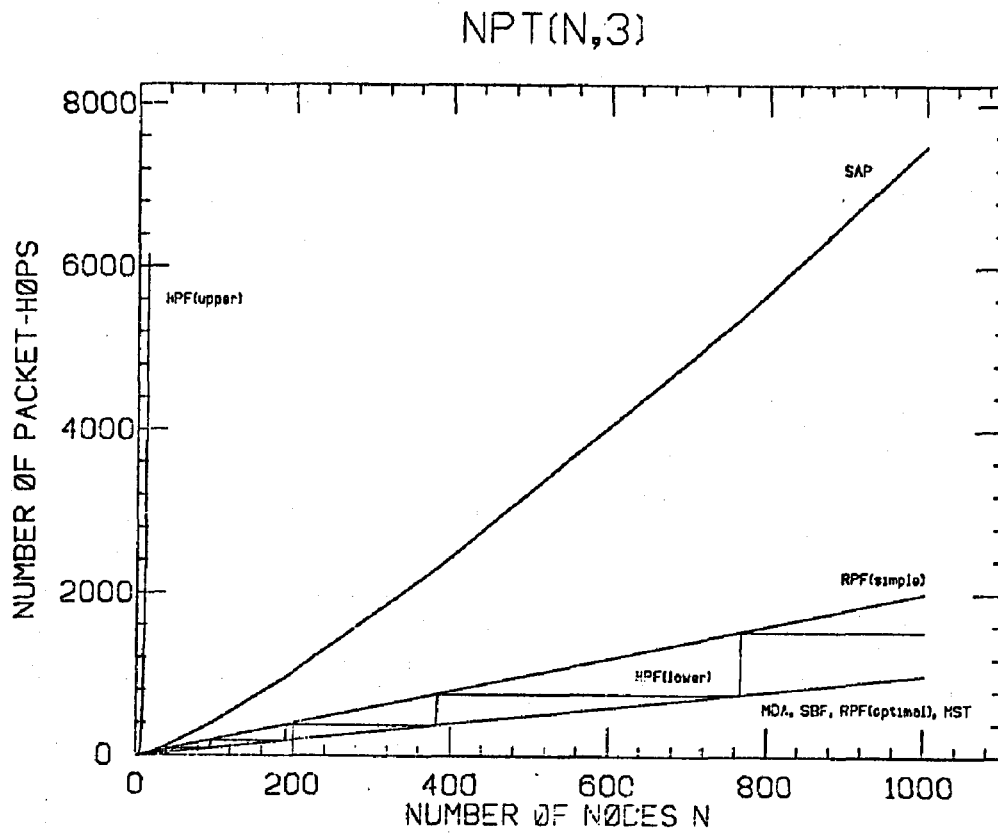


Figure 4.18. LOWER BOUND ON THE NUMBER OF PACKETS TRANSMITTED FOR REGULAR GRAPHS OF DEGREE 3 USING THE BROADCAST ROUTING ALGORITHMS.

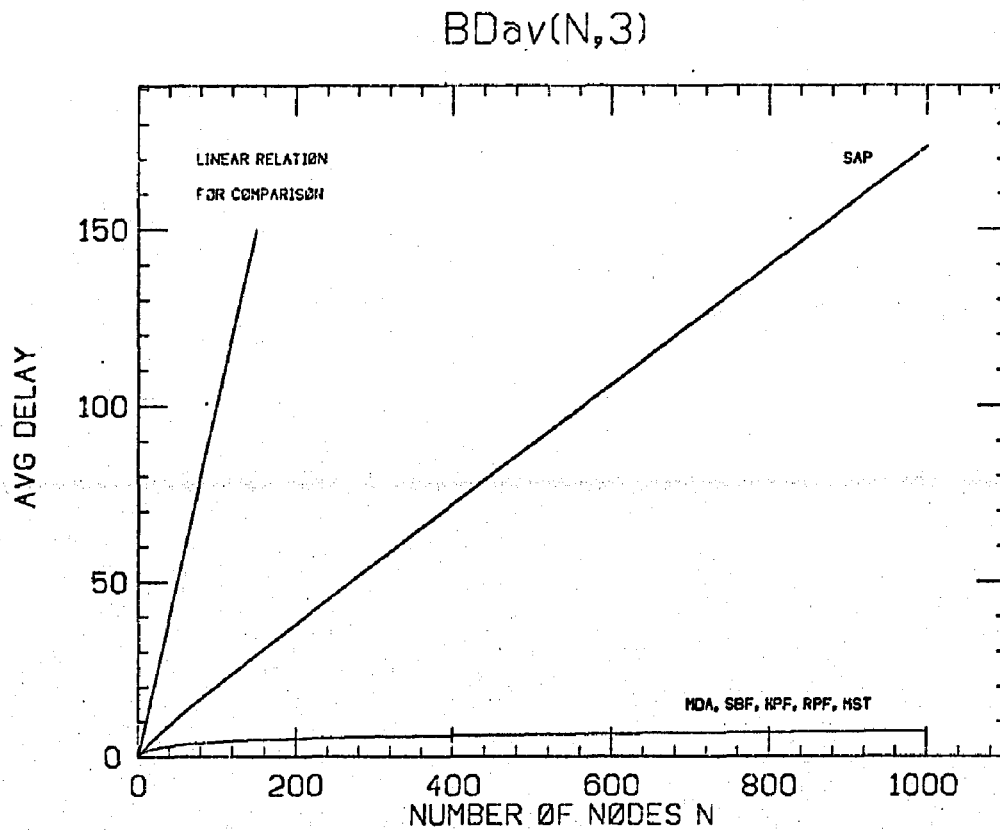


Figure 4.19. LOWER BOUND ON AVERAGE BROADCAST DELAY FOR REGULAR GRAPHS OF DEGREE 3 USING THE BROADCAST ROUTING ALGORITHMS.

$BD_{\max}(N,3)$

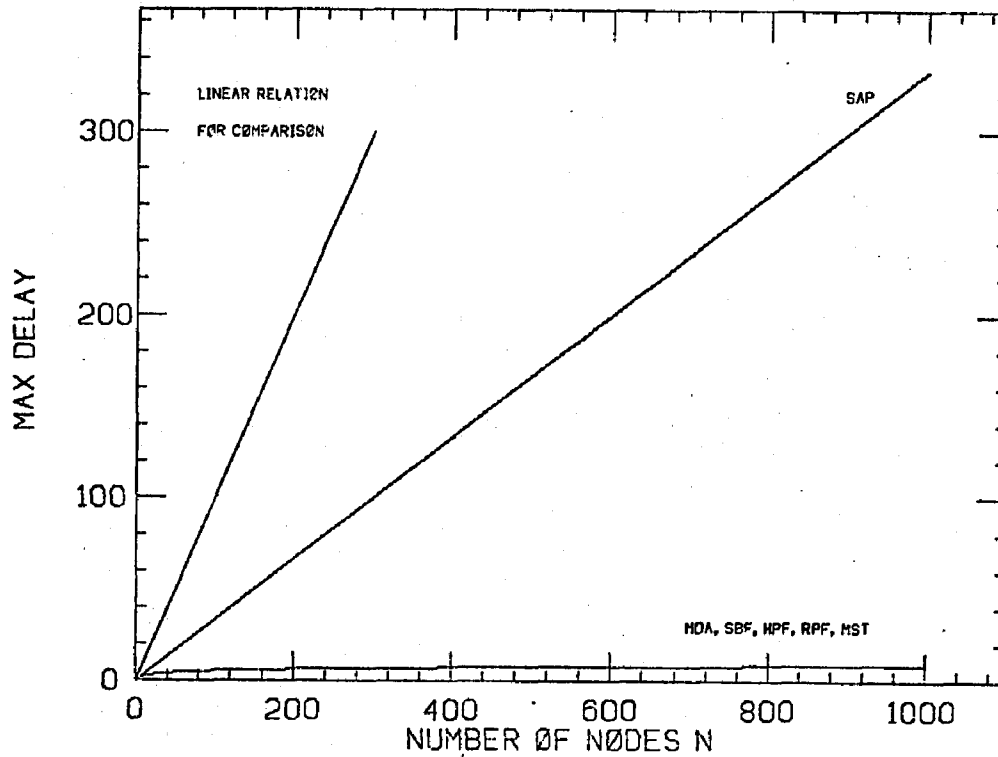


Figure 4.20. LOWER BOUND ON MAXIMUM BROADCAST DELAY FRO REGULAR GRAPHS OF DEGREE 3 USING THE BROADCAST ROUTING ALGORITHMS.

$BC(N,3)$

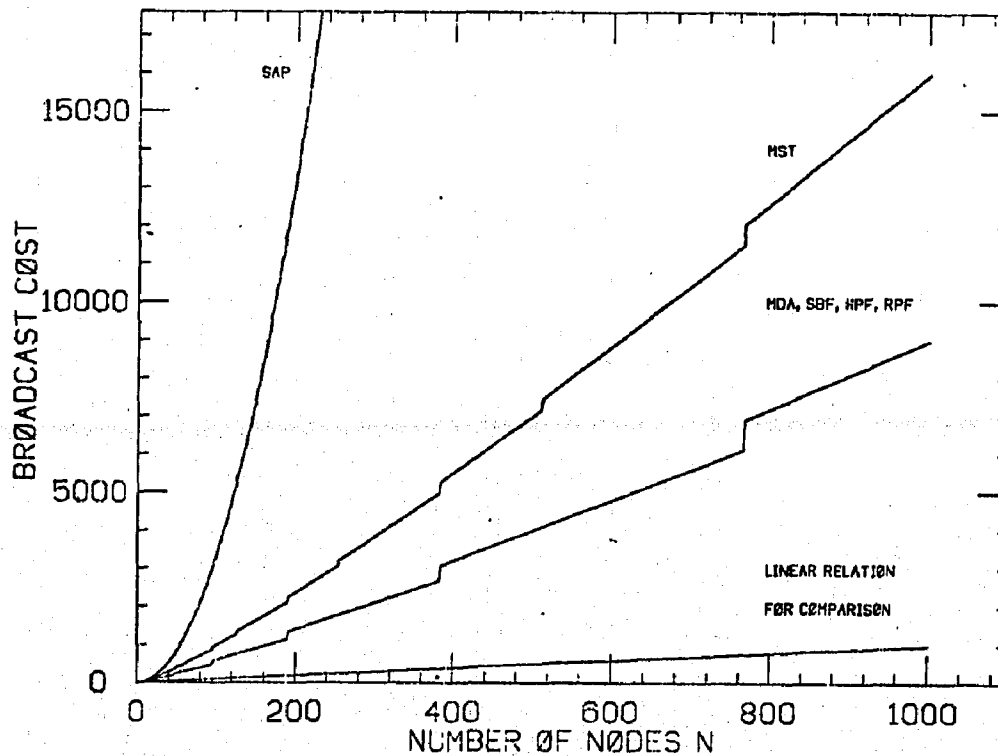


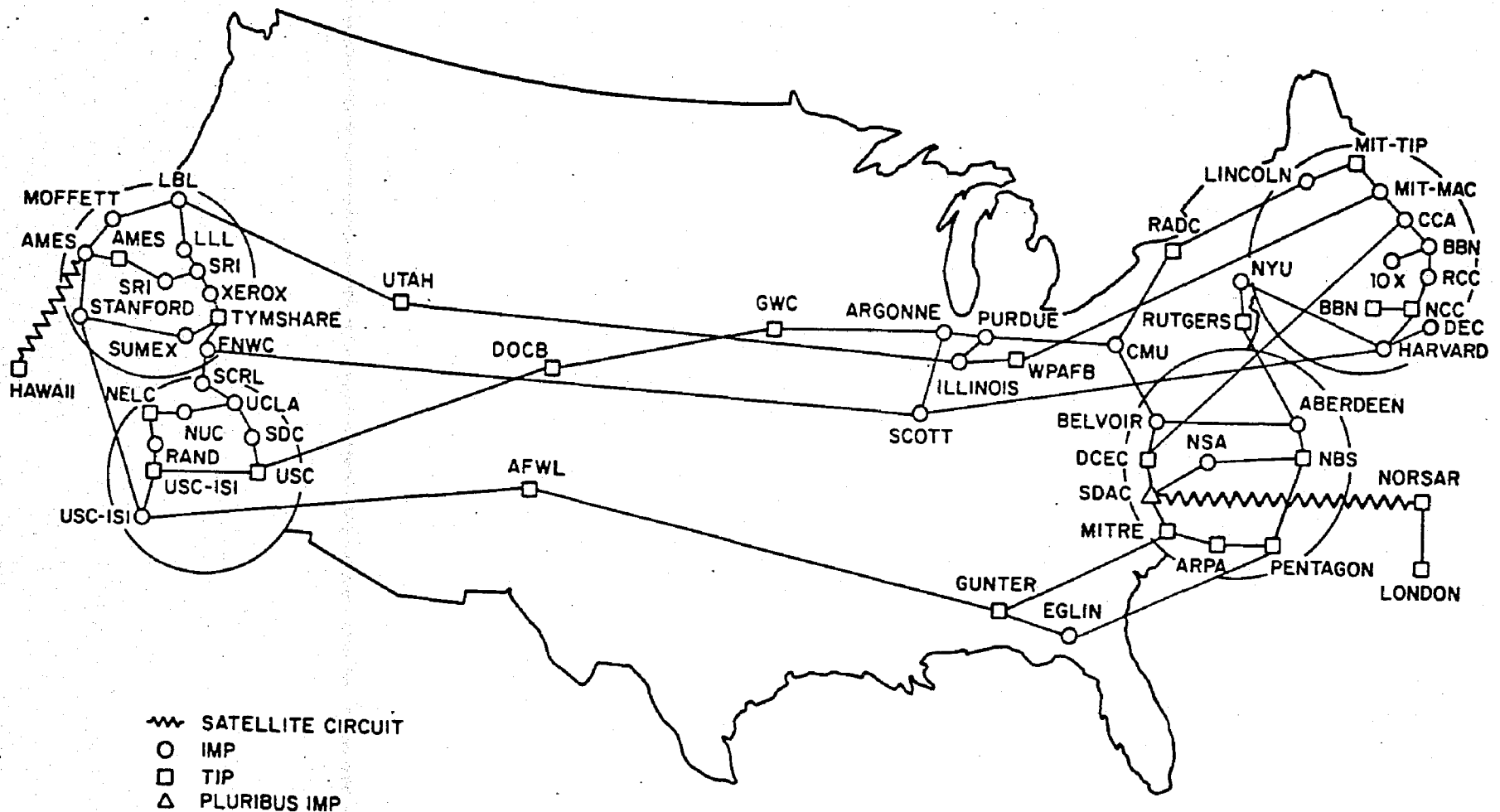
Figure 4.21. LOWER BOUND ON THE BROADCAST COST FOR REGULAR GRAPHS OF DEGREE 3 USING THE BROADCAST ROUTING ALGORITHMS.

4.10 Performance in the ARPANET

We now determine the performance of the different algorithms for the ARPANET topology. The performance measures are determined by taking the average of the measures as seen by each node. Figures 4.22 and 4.23 illustrate the geographic and logical maps of the ARPANET as of August 1976. Table 4.1 shows the values of the various performance measures. These values were determined using a computer program that computes, for all the algorithms, the performance measures as seen from each node for any graph, whose edge costs are known.

The maximum degree of any node in the ARPANET is 4, and the network has 59 nodes. Table 4.2 indicates lower bounds on the performance measures for a regular graph of degree 4 having 59 nodes. There is a large discrepancy between the corresponding entries of Tables 4.1 and 4.2 because the ARPANET topology does not resemble a regular graph and because the theoretical measures of performance are lower bounds. Notice that the values of NPT for the regular graphs when using hot potato forwarding represents upper bounds on NPT for the ARPANET, since the regular graph has more links than the ARPANET topology, and this is the factor that determines NPT.

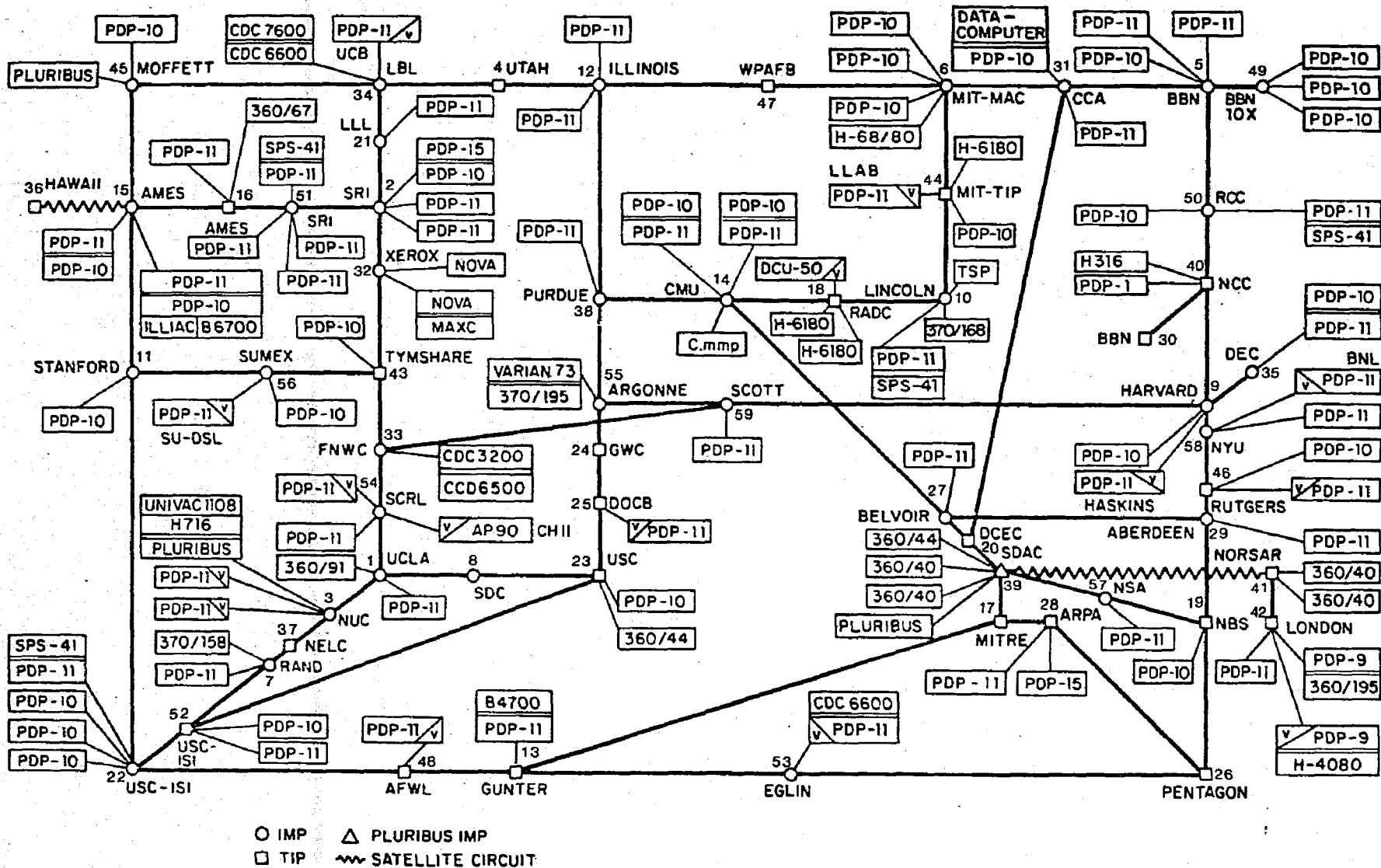
ARPANET GEOGRAPHIC MAP, AUGUST 1976



(NOTE: THIS MAP DOES NOT SHOW ARPA'S EXPERIMENTAL SATELLITE CONNECTIONS)

Figure 4.22.

ARPANET LOGICAL MAP, AUGUST 1976



(PLEASE NOTE THAT WHILE THIS MAP SHOWS THE HOST POPULATION OF THE NETWORK ACCORDING TO THE BEST INFORMATION OBTAINABLE, NO CLAIM CAN BE MADE FOR ITS ACCURACY)

Figure 4.23.

Table 4.1

ACTUAL PERFORMANCE OF THE ROUTING ALGORITHMS IN THE ARPANET

Measure Type	\overline{NPT} Packet- Hops	\overline{BD}_{av} Hops	\overline{BD}_{max} Hops	\overline{BC} Hops
SAP	330	20.6	34.5	2152
MDA SBF	58	5.3	9.1	538
HPF (upper) (lower)	* 618.6	5.3	9.1	538
RPF (simple) (optimal)	86 58	5.3	9.1	538
MST	58	12.5	25.6	1509

* Computation exceeded 30 min on IBM 370/168.

Table 4.2

PERFORMANCE OF ROUTING ALGORITHMS IN A
REGULAR GRAPH OF DEGREE 4 HAVING 59 NODES

Measure Type	$\overline{\text{NPT}}$ Packet- Hops	$\overline{\text{BD}}_{\text{av}}$ Hops	$\overline{\text{BD}}_{\text{max}}$ Hops	$\overline{\text{BC}}$ Hops
SAP	160	9.5	15	885
MDA SBF	58	2.8	4	236
HPF (upper) (lower)	* 160	2.8	4	236
RPF (simple) (optimal)	178 58	2.8	4	236
MST	58	2.8	4	377

* Computation very expensive.

4.11 Conclusions

In this chapter we have proposed some performance measures for broadcast routing in packet switched store-and-forward computer networks. We then determined lower bounds on these measures, for the different routing algorithms described in Chapter 3, by examining regular graphs.

This analysis provides a quantitative basis for comparing the broadcast routing algorithms. A network designer can determine the cost of performing broadcast in his network by using a computer simulation similar to the one used in this thesis for analyzing the ARPANET. Alternatively, the network can be transformed into a regular graph and then lower bounds on the performance measures can be determined.

The regular graph derived from the network topology could be one whose degree is equal to the largest degree of any node in the network, and whose number of nodes is equal to that in the network. This technique was used in section 4.10 on the ARPANET topology. Alternatively, the minimal regular graph containing the given network as a subgraph could be used. This construction was determined by Erdős and Kelly [Erdős73a]. Our experience has shown that both these techniques provide measures that are lower than the actual values. The suitability of such techniques as a useful heuristic that preserves the relative ordering between the measures for the different algorithms is a subject for future research.

CHAPTER 5

DISTRIBUTED FILE SYSTEMS: AN APPLICATION

5.1 Introduction

The file system is one of the most important and basic resources of a distributed operating system that permits resource sharing in a computer network. The extension of notions like integrating memory with the file system as in Multics [Daley68, Bensoussan72], or the dynamic association between memory and file pages as in Tenex [Murphy72], or that many files are executable programs, to a networking environment leads to the requirement that the computers within the network be homogeneous if resource sharing is to be successful. However, many files are in standard formats e.g. ASCII or EBCDIC, or can be restructured using a data reconfiguration protocol, making them suitable for use even in a heterogeneous computer network. For generality we will assume that the computer network consists of heterogeneous computers, but that the virtual memory seen by all of them is the same; the homogeneity being natural, or artificial via protocol translation (where possible of course). This chapter reviews and proposes models for the structure of a distributed file system in such an environment, and shows that an underlying broadcast communication capability greatly improves the performance of such a system. We have not solved all the problems, but propose a structure and a set of algorithms that we feel could be used to build a distributed file system.

No assumptions are made as to whether the computer network provides a single operating system, whose components are distributed, or whether the hosts in the network have independent operating systems designed to treat resources that are remote or local in a similar manner, thereby providing uniform access to all of them. The term distributed operating system will encompass both structures. Examples of distributed operating systems that also support distributed file systems are the Resource Sharing Executive (RSEXEC) in the ARPANET [Thomas73], the National Software Works (NSW) in the ARPANET [Crocker75, COMPASS76], and the Distributed Computing System (DCS) at the University of California at Irvine [Farber72a, Rowe75]. David Boggs at XEROX PARC has also been working on distributed file systems, and file systems suitable in a networking environment in which pages of files can be accessed over the communication subnet [private communication January 1977].

For the purposes of this thesis, a file is an organized collection of elements, which could be words, characters, or bits. The system which controls the mechanisms for access, creation, modification and deletion is called the file system. The collection of directories that control and provide access to the files is the catalog for the file system. Since directories are files, the catalog is a collection of files. At this level a file is formatless and is referenced by a symbolic name. Daley and Neumann provide an excellent discussion on the structure of a general purpose file system [Daley65]. The logical structure of the file system discussed in this thesis is based on their model.

A distributed file system (DFS) is therefore one whose components or resources (the files) are distributed among a number of computer systems, on whose secondary storage systems the resources reside. There are many advantages of having a DFS and of building distributed data bases [Booth72]. One of the primary reasons is to provide a fail-soft file system.

There are two aspects of the DFS that we wish to model:

(i) Mechanisms for structuring the directories of the file system, and search algorithms for locating a symbolically referenced file. The file, or subsets of it, can then be transferred to the primary memory of the system on which the request was originally generated. The structure of the directories must provide the desired level of access protection.

(ii) Algorithms for permitting files to migrate from one host in the distributed operating system to another so that all the files are (nearly) optimally located, based on an objective function that minimizes the overall cost for accessing and storing the files.

If the DFS permits only single copies of files, then file migration increases the efficiency with which files are accessed, but not the availability of files. If multiple copies are permitted, then the availability of files also increases.

For the purpose of keeping the problems and analysis tractable, certain simplifying assumptions are made about the files in the DFS.

Initially, it is assumed that there exists only one copy of any file in the entire file system, and that the operations permitted on the file will include creation, deletion, modification, examination and execution. It is assumed that mechanisms, similar to ones prevalent in non-distributed file systems [Murphy72, Madnick69], for controlling multiple simultaneous access are also present. Next, the existence of duplicate copies of read-only files will be considered, and finally the problems introduced by the existence of duplicate copies of modifiable files will be examined. It is also assumed that a file is accessed by moving the entire file from the file system storage into the memory hierarchy that simulates a user's virtual memory space. This latter constraint can be removed when appropriate cost measures for using portions of a file over extended periods of time are found.

Resource sharing environments should often provide transparency of location to the user, who might wish to be unaware of the distributed nature of the system. Resources which are remote, though referenced identically to locally resident ones, may take a longer time to become available. As a consequence, most host operating systems, e.g. those in the ARPANET [Roberts72, McQuillan72, Crocker72, Metcalfe73], are unaware of the topological structure of the communication network since the hosts are "users" of the communication network. However, when a distributed operating system is attempting to optimize use of its resources, knowledge of the topology (if the network is not inherently broadcast in nature) is essential. This will be readily apparent when algorithms for causing files to migrate between computer systems are considered.

No detailed assumptions are made as to the technology or structure of the distributed environment. The communication subnet could be packet-switched, circuit-switched or a multiaccess channel. The terminology used in this chapter will have counterparts in all communication subnets, though our analysis of file migration is biased towards store-and-forward networks. A host is a computer system that is a potential user and/or supplier of resources in the distributed operating system. A user is a person or a program that interacts with the host. A user process is the process associated with a user. A switching node is a device, in many cases a small computer, that accepts data and control from the host and sends it over the communication links, with the possible cooperation of other switching nodes, to the destination. The collection of switching nodes and communication links is the communication network or communication subnet. We assume that all communication between hosts is viewed as interprocess communication, and that it can be performed reliably [Cerf74, Cerf74a, Sunshine75]. Figure 5.1 illustrates such a distributed network environment.

One of the primary goals of the DFS presented in this thesis is to perform the necessary functions using distributed algorithms, under the assumption that no centralized information source or point of control exists. Such an assumption is necessary in order to preserve the reliability of the DFS. There are usually many points of control in a distributed algorithm, and so if any should go down then it is usually possible to continue to function (possibly in a degraded fashion). We will investigate the extent to which conventional algorithms used in implementing file systems can be distributed.

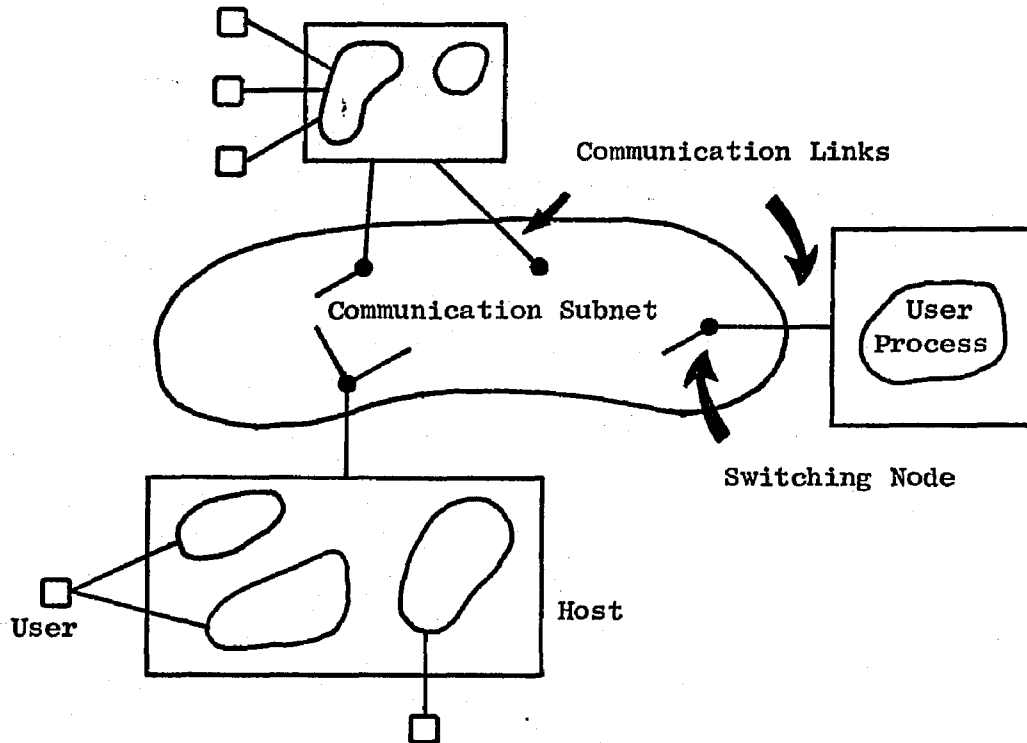
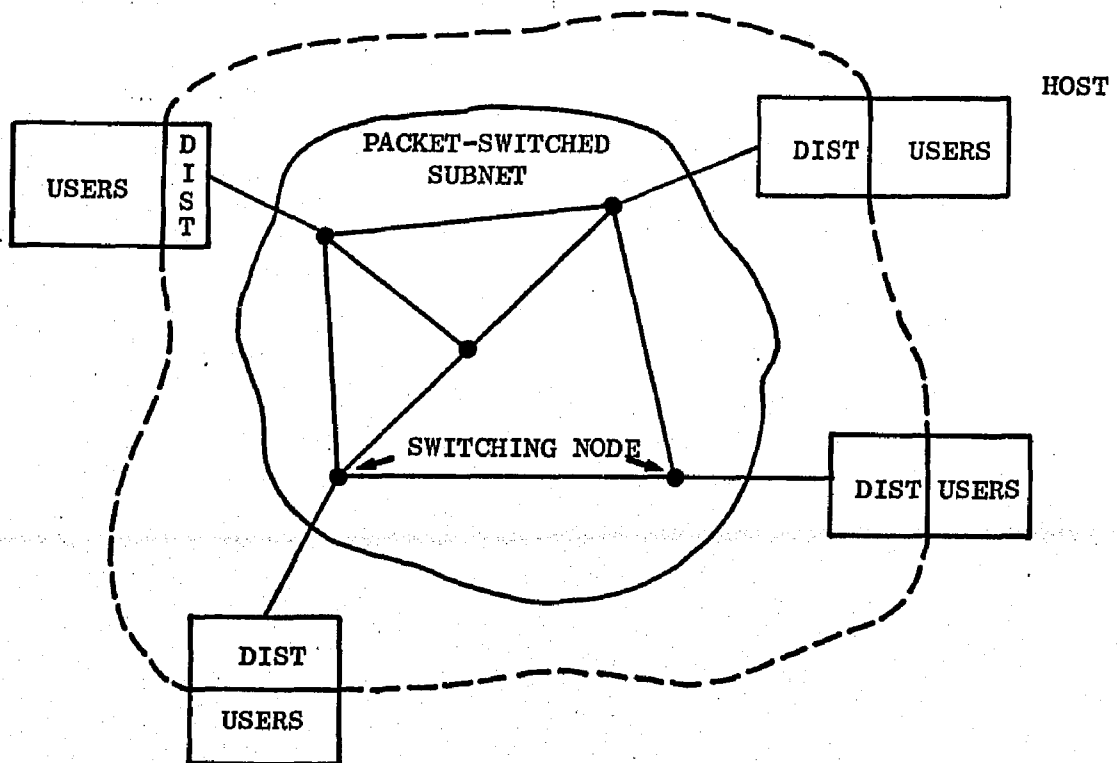


Figure 5.1. A DISTRIBUTED NETWORK ENVIRONMENT.



DIST \equiv LOCAL FILE SYSTEM RESOURCES THAT ARE PART OF THE DFS

Figure 5.2. A DISTRIBUTED FILE SYSTEM.

5.2 User Interface and Catalog Structure

The DFS will consist of local file systems at each of the hosts. Their resources are available to all users. Figure 5.2 shows this model. In general, the host may divide its local file system into two parts; a private part and a sharable part. The host may use the private part to provide storage for permanently resident files for its local users. It may even copy files from the DFS into the private file system, in order to have a permanently resident copy outside the domain of the DFS. Such copies may become inconsistent with their counter-parts in the DFS. The file system provided by the NSW consists of a sharable global NSW file space, as well as a private, non-sharable local file space at every host [Schantz76]. We do not make use of any such techniques in this thesis, and therefore assume independence between these two parts, and so leave the private part out of all subsequent discussions.

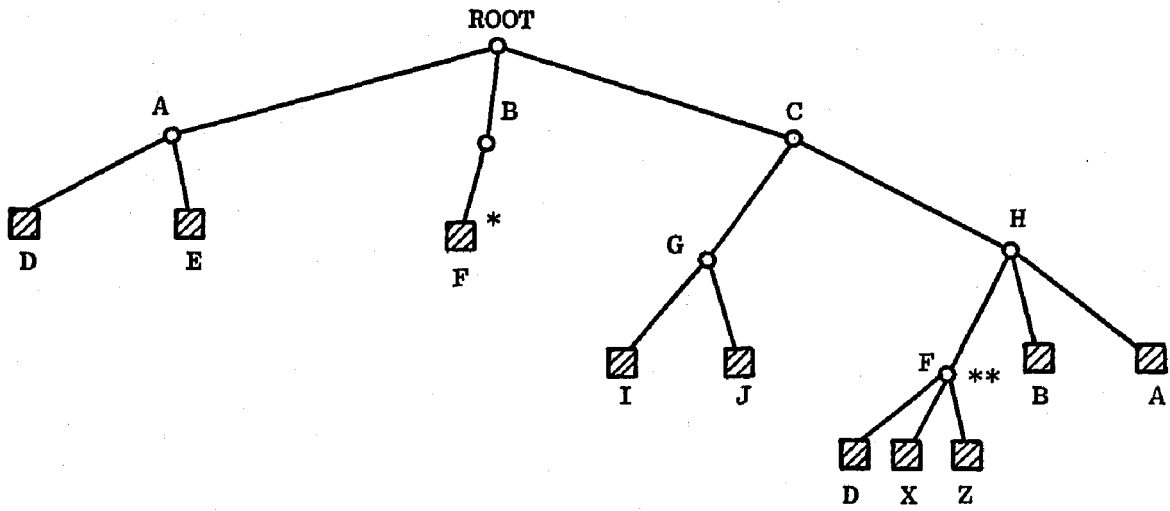
We will assume that the entire file system will have a hierarchical structure, since it seems most appropriate from the users' point of view [Daley65]. Files will be referenced symbolically and shared by a number of users. File names must be unique and must not change even when files migrate to other hosts, so that this movement is transparent to the users of the files. The file structure may be thought of as a tree of files, some of which are directories. Except for the root directory, each file finds itself pointed to by exactly one branch in exactly one directory. The tree name of a file will be its name relative to the root directory. Figure 5.3a illustrates such a structure. Links may

now be superimposed on this structure, such that files may be accessed from directories other than those present in their respective tree names. Note that tree names do not contain any links. The path name of a file is its name relative to the root directory and may contain links. Hence, in general, a file can have only one tree name but many path names. The tree name and path name of a file can be specified relative to a working directory other than the root. Since the tree name of the working directory is known the absolute tree name of the file can always be determined. Figure 5.3b illustrates a directory structure with links. The file systems of Multics [Organick72], Unix [Ritchie74] and Tenex [Bobrow72] have similar structures. In a conventional monolithic file system, the directory files in the catalog provide a reference point for naming files, access control to the files that are their offspring, and indicating where on secondary storage they reside.

We now consider different ways of physically structuring the catalog, and the assumptions and constraints required, in order to create a distributed system with the same logical structure as described above. Once files have been created and entered into the DFS catalog they may migrate and physically reside at any host. When a file moves, neither its tree name or path name changes, only its location.

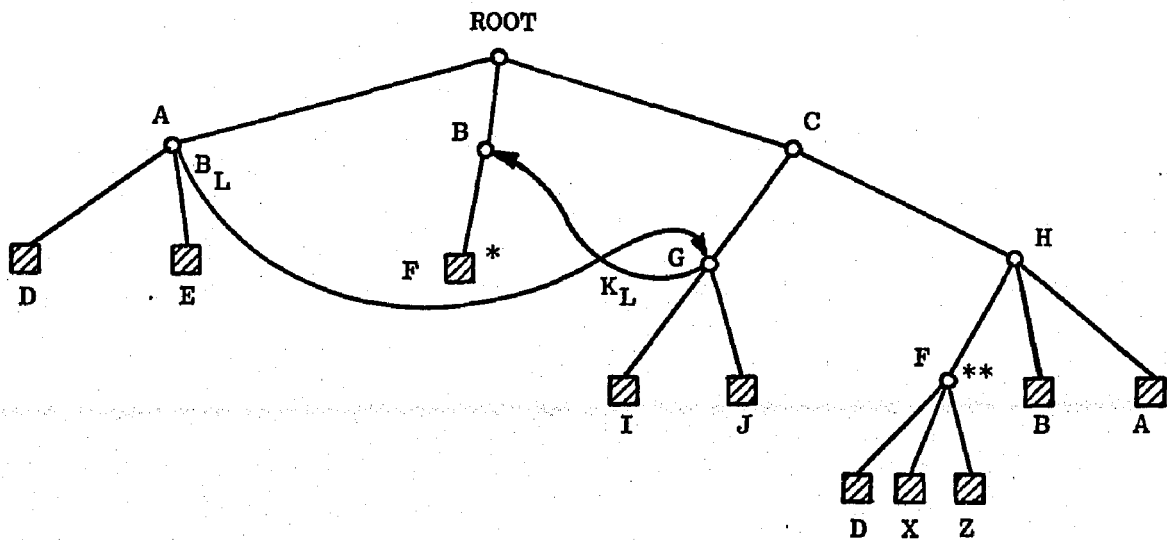
5.2.1 Centralized or Duplicated Catalogs

The logical catalog structure could be centralized at one host, with the non-directory files scattered among various hosts. Without getting into the implementation details, note that each host (except the



Tree name of file marked * relative to root is B.F and of file marked ** is C.H.F.

Figure 5.3a. A HIERARCHICAL CATALOG STRUCTURE WITHOUT LINKS.



Tree name of file marked * relative to root is B.F; path name of * relative to root C is G.K.F and, relative to the root, could be A.B.K.F.

○ Directory file; ▣ Nondirectory file; Links are subscripted with L

Figure 5.3b. A HIERARCHICAL CATALOG STRUCTURE WITH LINKS.

one that has the catalog) that has non-directory files, must have a private directory indicating where on secondary storage the files physically reside. This information might have been included in the centralized catalog, but such a restriction couples the various hosts very tightly, and prohibits each local file system from making any local modifications to the position of the files on its secondary storage, without updating the appropriate directory files. The global file space provided by the NSW consists of a centralized catalog resident on the Works Manager Host, with the non-directory files scattered among the Storage Hosts [COMPASS76, Muntz76]. We rule out this alternative since it contradicts our fundamental requirement (providing a fail-soft DFS system), since if the host with the directories goes down so does the entire DFS. Further, reference to any file requires network communication.

In order to overcome some of these problems, the catalog could be duplicated at a few or all hosts. This appears to be the ideal case, since files can be found quickly and the structure has sufficient redundancy built in it to be fail-soft. Such a scheme is, however, wasteful of space, and introduces a lot of complex problems. Since catalogs consist of files, there are now duplicate copies of many files. Directory files have the read-write property, and so how are the various catalogs kept consistent when non-directory files are created, are deleted, or migrate? If the catalog is duplicated at every host, then there is no need to have a separate private directory for locating the physical secondary storage location of a file, since the catalog at

every host can include this information, as in some sense the catalog is private to every host.

5.2.2 Partitioning the Catalog Based on Pointers

Rather than impose constraints on the position and movement of directory files, we would like the DFS to permit any file to migrate; should it be necessary. We now propose the structure of a DFS that permits this. It is very similar to a conventional monolithic file system, except that the physical location of any file (as determined by examining its parent directory file) is limited just to the identity of the host on which it currently resides.

Assume that there is only one copy of every file in the file system, and that it can reside at any local file system. Each directory file will point to the host on which its offspring reside. Of course, each local file system must have a directory that associates the tree name of a file, resident there, to its physical secondary storage location. Such a directory must have an appropriate data structure that permits a fast search by tree name for such files [TENEX-4]. We will call this data structure the Local File Directory (LFD). This "directory" is not part of the DFS, and is private to each host. Such a scheme does not preclude a file from migrating, as long as the file's parent directory is updated to reflect this movement. Figure 5.4 illustrates how a catalog would appear after some of the files had migrated.

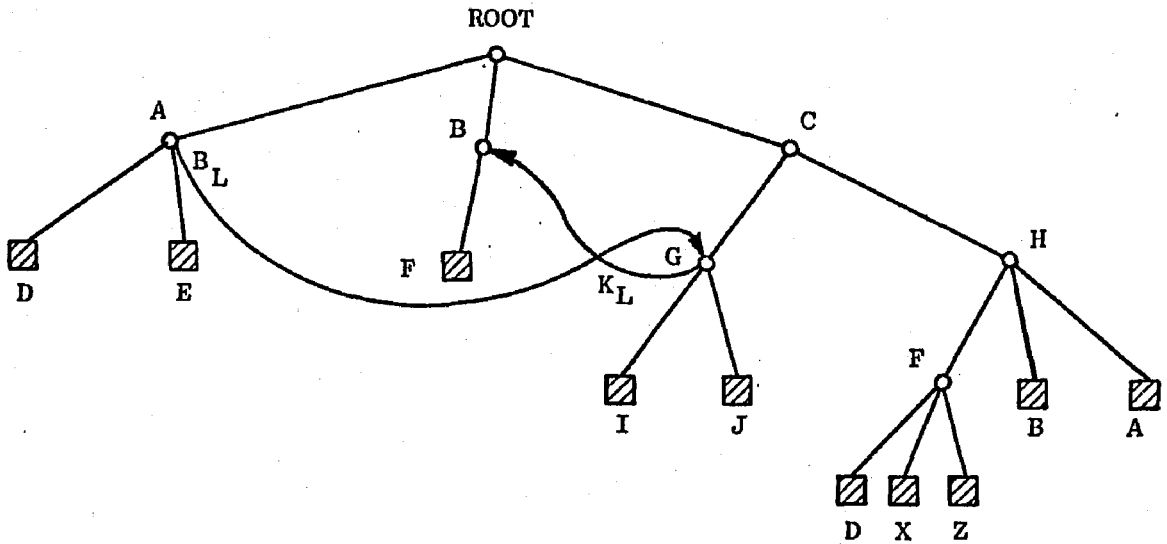
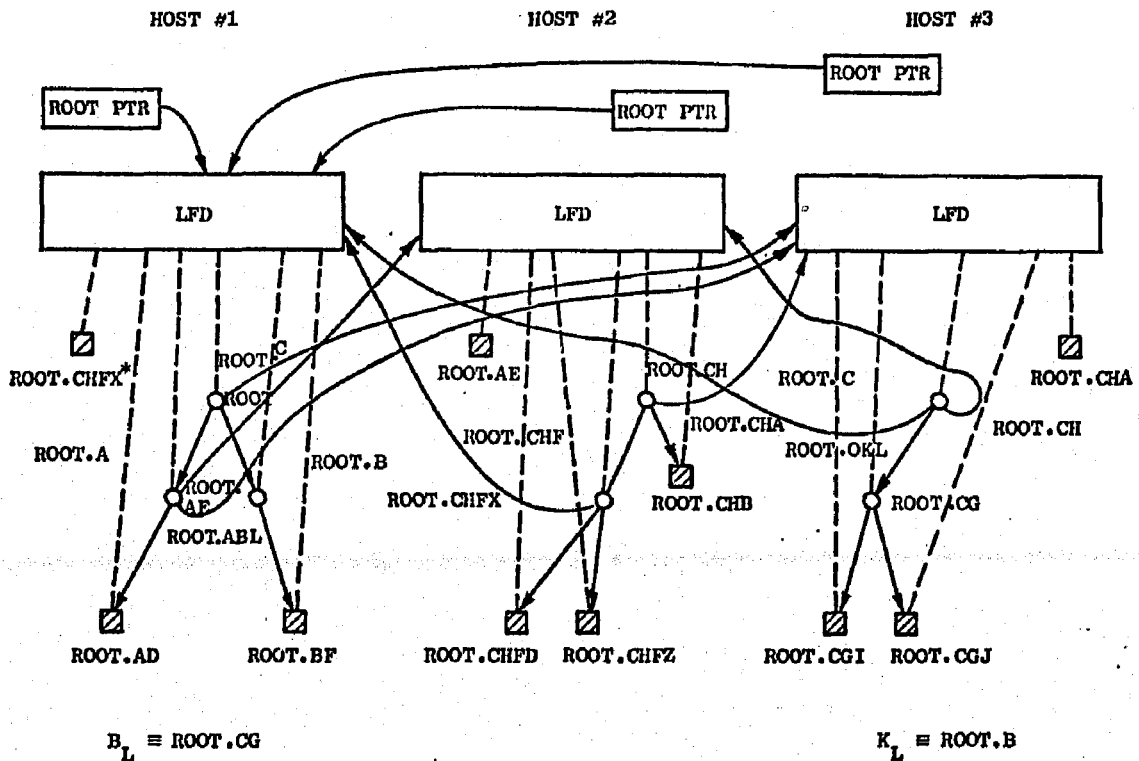


Figure 5.4a. A LOGICAL CATALOG STRUCTURE



○ DIRECTORY FILES; ◻ NONDIRECTORY FILES; LINKS ARE SUBSCRIBED WITH L; → POINTER TO APPROPRIATE HOST
 * Dots are left out of the file name past the root for pictorial clarity, since it introduces no ambiguity.

Figure 5.4b. PARTITIONING BASED ON POINTERS IN A DFS WITH 3 HOSTS.

When a file is referenced, the root directory must first be located in order to find the first directory file. Each of the directories in the path name, which may include links, is stepped through (as in a conventional monolithic file catalog), with the access rights being checked, and the location (host) of the next file in the path name being determined. The next file in the path name is found because the host on which it resides is known, and that host knows where on secondary storage the file resides by looking up the tree name in its LFD. When a file is being created, the same sequence of events has to be undertaken in order to determine whether the file can in fact be given that name. If it can, then the necessary directory files must be updated. The catalog files of the DFS are distributed in space, and so the search for a file is also distributed in space. Such a search can be implemented by having the locus of computation move from one host to another, until the file is found and transferred to the initiator of the search, or by having the host initiating the search interrogate each of the files constituting the path name, either by retrieving each of them into its own virtual memory for examination, or by interrogating the remote data base through an appropriate protocol. We elaborate on these two approaches, since they determine an implementation of the DFS, and structure of the directory files themselves.

If the files comprising the path name of a file being referenced are always looked at by the host making the request, then directory files must never have any information in them that relates to some specific local implementation, i.e. they must be formatted identically

and not have any information like local disk addresses. If the remote files are interrogated through an appropriate protocol, then such an implementation does not require the requesting host to examine directory files that are not resident at its local file system. As a consequence, the LFD at a host can use some of the directory files of the DFS for locating files on secondary storage. For example, if a file and its parent directory are both present at the same host, then the parent directory could point to a secondary storage location for the file. This is just an implementation detail. Directory files must be marked as "open", when in use, so that they are not modified inadvertently. For example, if a file and its parent directory were present at the same host, and that file was to migrate, then that file can only migrate after the parent directory has been updated appropriately. Such implementations can be made to achieve some degree of reliability when performing the distributed search, since the initiator of the search has control at all times. For example if a file with tree name A.B.C was being referenced, then the host referencing the file would have to examine the root directory, then A, and finally A.B before A.B.C was made available.

Alternatively, the requesting host could send a message to a process residing at the same host as the root directory, and have it send a message to a process residing at the same host as the first directory file in the path name of the file being referenced. The access rights would be verified and the process of sending messages would recur until the file was found. The file could then be

transferred into the virtual memory of the host initiating the request, and all the processes spawned along the way notified, so that they could disappear. Such an implementation can also be made to perform the distributed search reliably, since should it fail at any point for any reason, status information can percolate through the process chain to the initiator. Such an implementation does not require a host to examine or interrogate directory files that are not resident in its LFD, since control is transferred to a remote process to continue stepping through the directories in the path name of the file.

Such a catalog structure permits the easy extension of a monolithic catalog into a distributed one. However, every file in the path name of a file must be examined, before it can be accessed. This can be time consuming if the files are scattered around. The system is fail-soft in only one respect. If parts of the catalog are unavailable, then those files that do not have any of the unavailable files in their path names, can still be referenced. Of course, if the root directory is unavailable, then the entire file system is unavailable. Further, when files migrate it is necessary to update the parent directory. This may not be possible if the network suddenly becomes partitioned, or if a host goes down.

5.2.3 Partitioning the Catalog Based on Pure Broadcast

When partitioning of the DFS catalog is based on pointers, a directory file explicitly points to a host where its offspring reside. This feature makes it possible to locate a remotely resident file very

easily, but degrades the reliability of the DFS, since there is a single copy of the root directory, and file pointers must be correctly updated.

An object may be searched for, either by following a set of pointers, or by performing an associative search. We now propose a catalog structure that uses broadcast protocols for performing a distributed associative search to find the host where a file resides. This catalog structure is an extension of the one that used pointers to locate a file.

The root directory is vital for locating every file. Normally every local file system would have to know where it was resident, in order to locate the first file of a tree name. Each local file system, however, knows which files are present at its site by examining its LFD. These include files at level one in the logical catalog, and so it is possible to do away with the root directory, as we shall see. In the scheme to be described, when a file does not reside at the same host as its parent directory, the parent directory does not point to the host where the file resides. The directory simply reflects the fact that it does not know who has it. The catalog would be partitioned in a manner similar to that illustrated in figure 5.4b, except that the root, root pointers, and inter-host pointers would not be present.

The search algorithm is essentially the same as the one that used pointers, except that a broadcast search protocol is used to determine the location of a file instead of following inter-host pointers. The search could be implemented with the locus of control moving from a

process on one host to a process on another, or with the control being exercised from the host initiating the request. In the latter case, the LFD should first be examined before initiating the broadcast search. For example, if host 1 make a request for a file with path name A.B.C, and both A and A.B.C are present at host 1, but A.B is present at host 2, then after host 1 has examined A.B (explicitly or implicitly), it should first examine its LFD for A.B:C before initiating a broadcast search for it!

When a file is being searched for, a message must be broadcast for every file in the path name whose location is unknown. A local file system responds to a broadcast search message by examining its LFD and complying with the details of a specific broadcast protocol and the distributed search protocol.

In general, such a catalog partitioning permits as general a file system as proposed by Daley and Neumann, if each of the directories in the path name is stepped through until the file is found. Such a scheme does not preclude files from migrating. This scheme is more resilient to failures than the one based on following pointers, because all requests do not have to go through a critical resource; the root directory, and when files migrate pointers do not have to be made consistent. The increased reliability is bought at the expense of increased communication. Such a scheme is not suitable, in general, in networks where broadcast routing is expensive, and the path name of a file can be arbitrarily long, since the process of interrogating the constituent directories may take a long time. Chapters 3 and 4 have

examined different ways of achieving broadcast routing in store-and-forward packet switched computer networks. The relative performance of these techniques has also been determined in these chapters.

Farber and Heinrich propose a scheme for structuring the file catalog in the DCS [Farber72b]. Their scheme is very similar to both the schemes based on pointers and pure broadcast (cf. section 5.2.2 and 5.2.3). This comes about because communication in the DCS is based on process addressing rather than host addressing, processes can migrate, and the communication subnet is essentially a multiaccess channel.

The catalog is partitioned into a number of units that contain one or more directories. In order to find a non-directory file, a broadcast is first performed to find the identity of the process that knows something about the user's master directory. This process is then interrogated to find the identity of the process that knows the identity of the next process in the hierarchy. This is repeated until the file is found. Since processes can migrate the messages must be broadcast to all hosts. In general, broadcast is performed as many times as the depth of the path name of that file. Figure 5.5 illustrates such a catalog. The number of broadcasts necessary to find a file in this example is three. Such a technique is simple and appropriate in the DCS, whose subnet is a multiaccess channel and where network addressing is based on process names. Such a catalog structure, in general, permits links.

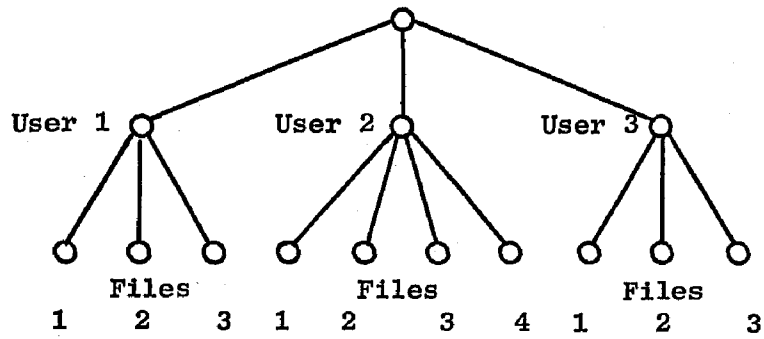


Figure 5.5a. LOGICAL STRUCTURE OF THE FILE SYSTEM IN THE DCS.

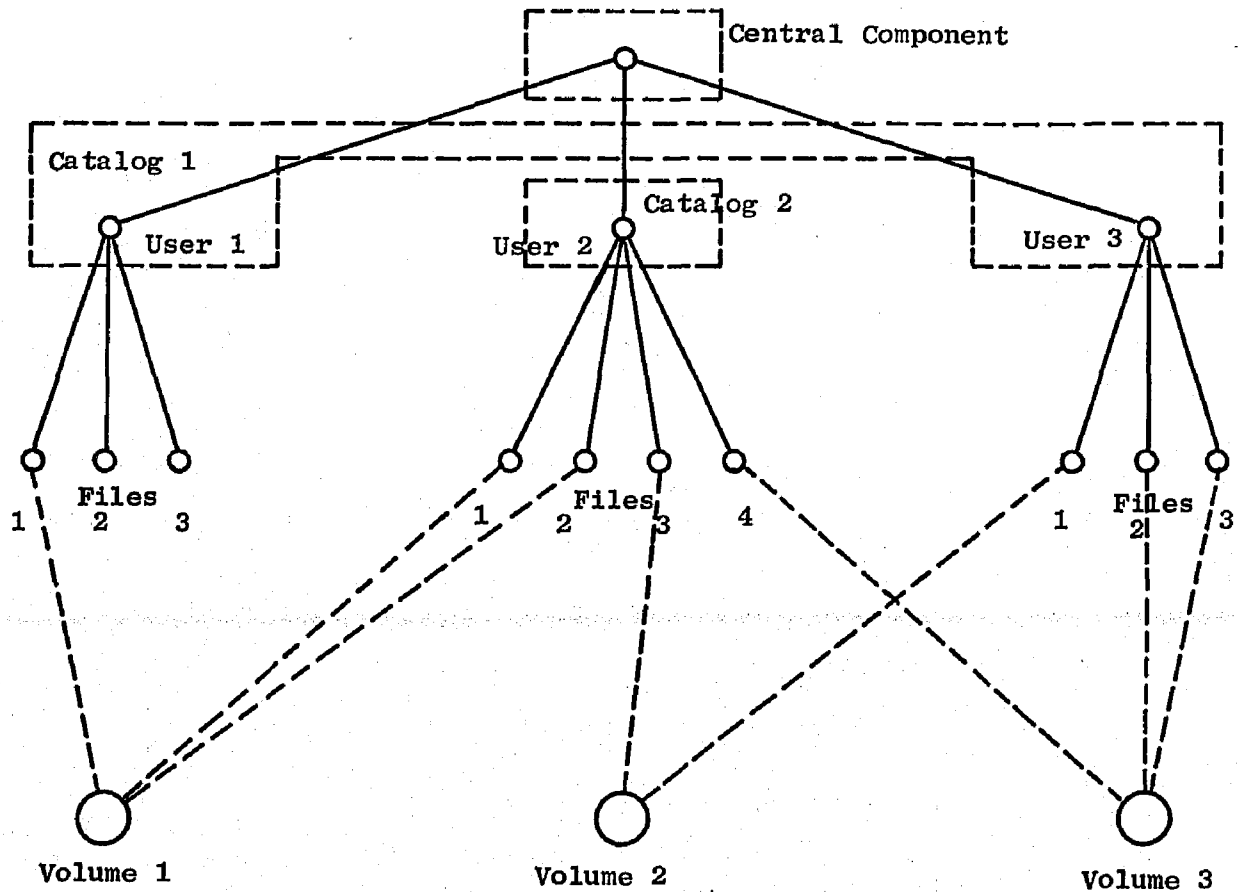


Figure 5.5b. DOTTED LINES SUPERIMPOSE A POSSIBLE PHYSICAL ORGANIZATION.

(Figure adapted from [Farber72b])

The DCS achieves a fail-soft file system since it treats the file system as a single level, instead of the hierarchical structure in say Multics. The first two levels are primarily for narrowing down the associative search any process would have to perform. The access protection checks are made while attempting to reference the desired file and not at intermediary directories. Further, all files have their path name associated with them, should it be necessary to find a file without having to go through the directories. We believe that these two are good ideas. For these reasons, however, it appears that the file system in the DCS does not permit links. We believe that aliasing can still be provided using a single level file system, and explore this possibility in section 5.2.5.

5.2.4 Partitioning the Catalog Based on File Usage Patterns

All files can be found using the broadcast search described in section 5.2.3. The amount of information in the catalog is necessary and sufficient for the DFS to work, although rather inefficiently if the proportion of requests to remote files is large. The performance of this scheme can be improved, if some additional information on file usage patterns is kept by each of the local file system. This information might help locate a file without having to perform a broadcast search in many cases.

In addition to the LFD every local file system should have another directory (structured similar to the LFD) that contains what it believes are the current locations (host identities) of some recently used

non-resident files. This will be called the Recently Used Directory (RUD). For a local file system to have an entry in its RUD, the file must have been resident at the host pointed to, though it may have since migrated. Hence, the important property of the RUD is that its contents need not be correct when it is examined, though it must have been correct at some time. The RUD is private to each host and not part of the DFS.

5.2.4.1 Search Algorithms

We now briefly indicate how this DFS structure would operate. When a file is referenced (and its path name is known), each of the directories in the path name must be examined in order to expand links, and to perform the necessary access protection checks until the desired file is found. As we described earlier, the distributed search for a file can be performed by transferring the locus of control from a process on one host to a process on another, until the file is found, or by letting the host making the request examine/interrogate each non-directory file itself. We describe how both search schemes would benefit from the RUD.

First, consider the case where the search is initiated and controlled from the host making the request. As in the case with pure broadcast scheme, the host continues to examine directory files until it does not know where the next one is located. Instead of performing a broadcast search at this point, the host could chain down RUD entries in various hosts, starting from an entry in its RUD, or maybe from the RUD

at the host where the current directory file was located. The choice of which RUD entry to start from could affect the outcome of the search. Such a search will either find the file (directory or non-directory), or reach a point where no more RUD pointers are available. This search will be called the cascade-search. If the search succeeds, then the file is examined, and the search for the desired file would continue if necessary. If the cascade-search fails, then the host must resort to the broadcast search. Of course, it might broadcast the search request only to a subset of all the hosts, since it thinks that some of them already do not have the file.

Now consider the implementation where the locus of control is transferred from host to host until the file is found. As in the case with pure broadcast, a point may come when the location of the next file in the path name is unknown. Instead of transferring the locus of control to a host through a broadcast protocol, control could be transferred to the appropriate host through the cascade-search. If the cascade-search succeeds the initiator of the cascade-search transfers control to the appropriate host. If the search fails, then the initiator of the cascade-search must transfer control via a broadcast search protocol. The broadcast message could be sent to possibly a subset of the hosts.

We prove in section 5.2.4.2 that, if the system functions correctly, (i.e. no tables have been corrupted) then the cascade-search terminates - it either finds the file or comes to a dead end. This means that it is not possible for the search to get caught in a loop.

We believe that files will not move around rapidly and so the above technique is appropriate. The size of the RUD at each local file system is dependent on available space and primarily affects local responsiveness. Note that replacement algorithms for the RUD must also be considered. They will be determined by file reference patterns, a subject that has received some investigation in [Stritter76].

The catalog structure and search algorithms presented here are an extension of the techniques used by the RSEEXEC. The RSEEXEC has a less general file system catalog than proposed here, and does not permit automatic file migration, and so the techniques employed there are simpler. The user profile in the RSEEXEC maps file names in a user's virtual file system to the physical location of the files in the ARPANET, and so resembles the RUD.

This discussion leads us to conclude that if files are to migrate, then the logical tree structured catalog, which may include links, is solely for the convenience of the user, who can name files relative to a given reference point, and for the file system to perform access protection checks at every stage. The file system may have to employ different mechanisms for searching for a file.

We examine the cascade-search in greater detail in the next two subsections, and then examine in section 5.2.5 the structure of a DFS that has restrictions on links in the catalog, but makes use of techniques presented in this section. This constraint reduces the excessive overhead of examining each directory file in path name of a

file. We then discuss some of the requirements of broadcast search protocols in section 5.2.6.

5.2.4.2 Properties of the Cascade-Search Algorithm

This section examines in detail the properties of the cascade-search described in section 5.2.4. A few observed axioms regarding the search algorithm are now stated.

Axiom 5.1: The RUD entry for a particular file at any host can only point to one other host; i.e. it is unique. The file must reside, or have resided at the host pointed to.

Axiom 5.2: When a file moves to a host, the RUD entry for that file at that host (should one have existed) must be cleared. Hence, a file can not both reside at a host and be present as an entry in the host's RUD.

Axiom 5.3: At any time, a host can reclaim the RUD entry for any file.

Axiom 5.4: When a file moves from one host to another, an entry is created for it in the RUD of the host where it originally resided, and the entry points to the new host. (This axiom is only included for the ease of proving properties of the algorithm, and is consistent with Axiom 5.3).

Under the conditions of a perfectly reliable system, the algorithm, when initiated from a host, is said to terminate (if the file is found or no further pointer exists) in a time proportional to the number of hosts in the DFS.

Theorem 5.1: The cascade-search terminates.

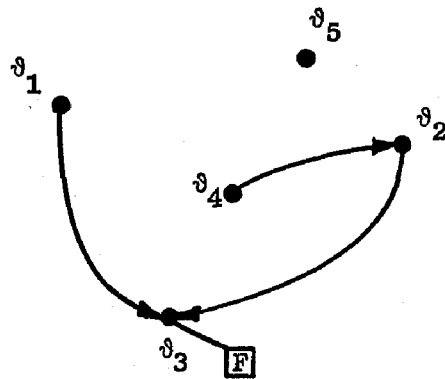
Proof: Since there is no interaction between the various files in the DFS, the theorem is proved for any one file F .

Let N be the number of nodes in the DFS. Let $a_{i,j}$ be a directed arc from node i to j if there exists an entry for F in the RUD at i , and it points to j . This is true for all $1 \leq i, j \leq N$, $i \neq j$. By definition arc $a_{i,j}$ has head i and tail j .

Let G be a simple directed graph, such that $G = [V, A(t)]$, where V is a finite set of vertices of cardinality N , and $A(t)$ is the finite set of arcs $a_{i,j}$ at time t . A path $p_{i,j}$ of G is a cascade-search sequence initiated at i and terminating at j . Figure 5.6 shows such a graph G .

From Axiom 5.1 it can be concluded that only one arc can leave any node. From Axiom 5.2 it can be concluded that any node that has no arcs leaving it represents a case where the file either resides at that node, or does not and the node does not know where the file is. Such a node is a point of termination for the algorithm. Since hosts may reclaim RUD entries at will (Axiom 5.3), points of termination may appear anywhere in the graph.

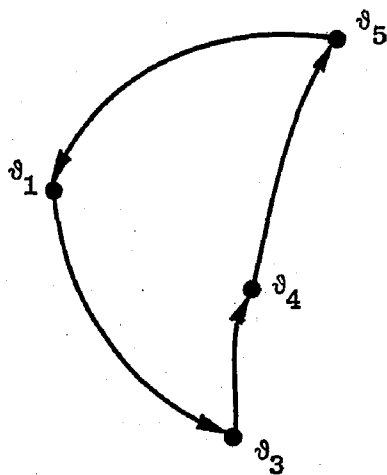
The theorem is proved by showing that G is always loop free, when files migrate, under the constraints of the four axioms. The theorem is proved by contradiction. We show that when a file moves, if a loop is created, then an axiom is violated.



$$V = \{v_1, v_2, v_3, v_4, v_5\}$$

$$A(t) = \{a_{1,3}, a_{4,2}, a_{2,3}\}$$

Figure 5.6. A GRAPH $G = [V, A(t)]$.



$$V = \{v_1, v_2, v_3, v_4, v_5\}$$

$$A(t) = \{a_{3,4}, a_{4,5}, a_{5,1}\}$$

$$A(t+1) = \{a_{1,3}, a_{4,5}, a_{5,1}\}$$

Figure 5.7. A LOOP IS PRODUCED IF AXIOM 5.2 IS VIOLATED.

Consider a G with $|V| \geq 1$ and $|A(t)| \geq 1$, such that the cascade-search terminates. The file may or may not be present at the tail of any arc. Now, if the file moves it may do so to a node which does not already lie on the graph, in which case an RUD entry will be created by virtue of Axiom 5.4, and the cascade-search will terminate since there is no path continuation from the new node. On the other hand, the file may move to a node already on the graph. By virtue of Axiom 5.2, the RUD entry (should one have existed) at the new node of residence must be removed. If this entry is not removed, and the node from which the file moved was also in the graph, the addition of the arc will produce a loop. Figure 5.7 shows a file moving from node v_1 to v_3 , and producing a loop if and only if v_3 does not remove arc $a_{3,4}$.

Since no loops are introduced when a file moves, the cascade-search terminates. RUD entries are created only if the file actually resides at that destination. Hence, in a correctly functioning system, there is no spurious way of introducing RUD entries, which may cause loops.

Q.E.D.

Corollary 5.1.1: If there exist duplicate copies of a file (with the same name) within the DFS, then the cascade-search still terminates.*

*Assume that the consistency problem has been solved for duplicate copies of modifiable files, or that the duplicate copies are read-only.

Proof: For the purpose of the proof, each of the duplicate copies can be thought of as a separate file, and since the proof of Theorem 5.1 assumed independence between files, the corollary follows.

Q. E. D.

5.2.4.3 Reliability of the Cascade-Search

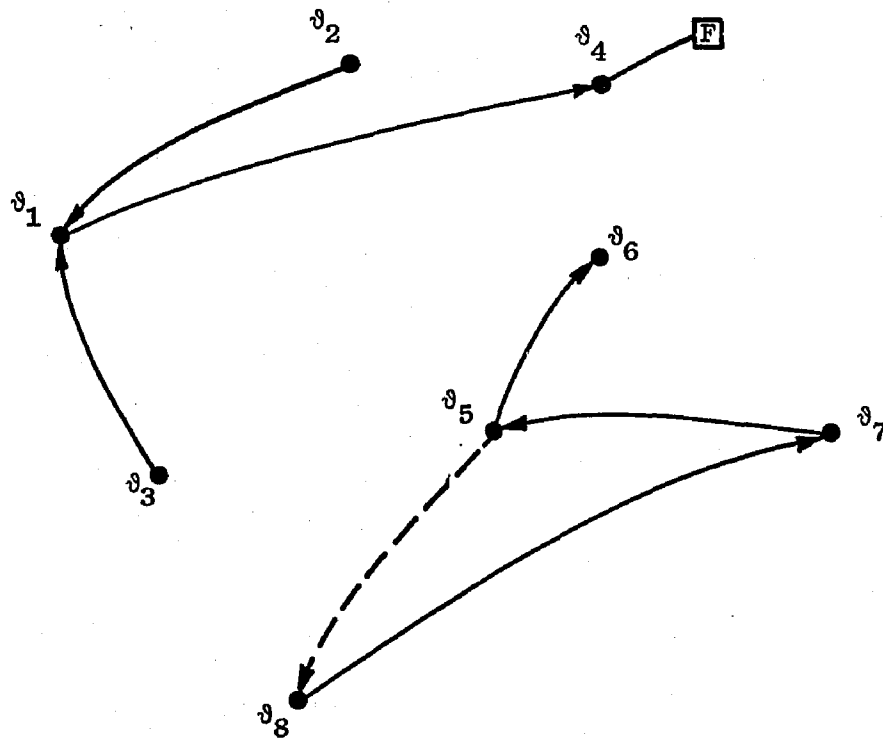
It was proved that the cascade-search will always terminate in a finite time under the assumptions of a correctly functioning system (no random errors to change table entries). In such a correctly functioning environment, the protocol used for implementing the cascade-search could be one in which the initiator of the search listens for a response on a particular port, the identity of which is specified along with the request for the file. The initiator is unaware of the number of RUD pointers chained through by intermediary hosts, and will either receive the file from the host that has it (and possibly update its RUD entry for this file), or realize that the cascade-search has failed, or timeout. The latter signifies that the file was most probably not found using the cascade-search and an alternative approach should be taken.

If the search is performed as described above, then the host from which the request was initiated, has no idea how many other hosts were involved in searching for the file. This is of no consequence if the file is found. In considering the action taken if the cascade-search fails, the requesting host will have to broadcast the request to all hosts. If it knew which ones were visited by the cascade-search, and

thus did not have the file (at that time), then the broadcast request need only go to the remaining hosts, thus minimizing the extra network traffic required to find the file. Of course, one can construct an example where a race condition causes a file not to be found. The file being searched for could have moved to a host that was visited by the cascade-search and at that time knew nothing about the file. The broadcast to the remaining hosts will not find the file, unless one of those hosts has an RUD pointer to the file.

On the other hand, if the DFS was unreliable, then the cascade-search could fail because of the existence of a loop in which a request circulates indefinitely, or because a request is lost. Figure 5.8 shows how the creation of a spurious RUD entry at node v_5 causes there to be a loop. $A(t)$ is the set of arcs that exist at time t , and $A(t+1)$ the set at time $t+1$. A spurious entry replaced $a_{5,6}$ by $a_{5,8}$ at $t+1$. A loop in the cascade-search has been created if it is initiated from v_5 , v_7 or v_8 for the file. If the requesting host knew which hosts were involved in the loop, it could attempt to break it.

The cascade-search need not be performed "recursively", but could also be performed "iteratively". That is, other hosts do not perform the search on behalf of the initiator, but instead each host on the search path returns the contents of its RUD entry to the initiator. Hence, the initiator iteratively interrogates each host on the search path until the search terminates. In the process of doing so, the initiator can keep track of each host visited, and thus can determine the existence of a loop. The loop can be broken by having all hosts



$$V = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8\}$$

$$A(t) = \{a_{3,1}, a_{2,1}, a_{1,4}, a_{8,7}, a_{7,5}, a_{5,6}\}$$

$$A(t+1) = \{a_{3,1}, a_{2,1}, a_{1,4}, a_{8,7}, a_{7,5}, a_{5,8}\}$$

Figure 5.8. SPURIOUS RUD ENTRY AT v_5 AT TIME $t + 1$ PRODUCES A LOOP IN THE CASCADE-SEARCH.

involved in the loop purge their RUDs, since it is not possible to determine the faulty host. Another approach to loop detection is to pass a counter with the recursive search. This counter gets incremented each time, and if it exceeds a certain count (the number of nodes in the DFS), then a loop exists. Breaking the loop requires more information to be passed around.

Note that a "recursive" or "iterative" implementation of the cascade-search can be used with either of the two implementations of the algorithm that finds the a file; i.e. having the locus of control move through the network, or having the host requesting the file do the interrogation of the directory files.

5.2.5 A DFS with Constraints on Links

The search algorithms described in sections 5.2.1, 5.2.2, 5.2.3 and 5.2.4, examined each directory file in the path name of the file being referenced. This was necessary because the tree name of the file being referenced was not known at the time it was referenced, and because access protection checks were performed when examining each directory file.

If the access protection for each file is stored in a file descriptor block that moves with the file, then access protection checks need not be made at every step in the search, but only when the file is found. Further, if the tree name of the file being referenced was known at the time it was referenced, then again there is no need to examine

the directory files in its name, in order to find the file. Hence, even though the logical file system is structured like a tree, for reasons concerning how a user wishes to view his file system, for purposes of protection and searching for a file, they can all be viewed as existing at the same level. This is how the distributed file system in the DCS is structured. The advantages of such a design is that files may be found relatively quickly, since excessive communication does not have to be performed in order to find all the directory files first. Further, even if directory files are unavailable, it is possible to find the referenced file. The disadvantages of such a structure is that file descriptor blocks may become large, and there can be large overheads when certain file operations are performed; for example when a file is created or deleted, both the file and its parent directory must be examined and searched for independently.

A one level file system can be achieved, if any of the following three approaches are taken regarding links and aliases in the naming of files. Each of these restrictions guarantees that the tree name of the file being referenced is known at the time it is referenced.

(i) Links could be prohibited entirely. Therefore the path name of a file corresponds to its tree name, which is unique.

(ii) There are no links in the catalog, but files can have aliases. The file descriptor block could contain all the aliases by which that file is known.

(iii) The only links permitted in the catalog are those in the working directory of the user. These links must point to the tree name of a file. Since the tree name of the working directory is known, the tree name of any file relative to this directory is also known. For example, if the tree name of the working directory is A.B, and this directory has a link called C to a file whose tree name is X.Y.Z, and a branch to a file called D, then reference to a file C or D relative to the working directory is all right, since they translate to tree names X.Y.Z and A.B.D respectively. A reference to a file C.E relative to the working directory would be improper since the tree name of such a file can not be determined without first examining X.Y.Z. Search for an improperly referenced file will fail.

We briefly indicate how the search technique described in section 5.3.4, the cascade-search, could be used in a single level file structure. When a file is referenced, its tree name is first determined. The LFD at the host where the request was made is first examined to see if the file resides locally. If so, the search is over. If not, the local file system assumes that the tree name is a proper file name, and proceeds to find it. It could perform a broadcast search, by which each local file system examines its LFD to see if the file is present, and if so communicates with the requesting host to determine if the requesting user process has the correct privileges. This is an expensive way to search for a file, and so the requesting host first initiates a cascade-search if there is a RUD entry for that

file. If the cascade-search finds the file, then the search is over, otherwise a broadcast search (to possibly a subset of the hosts) must be performed.

5.2.6 Broadcast Search Protocols

We have shown that broadcast search protocols can be used to find files, or any resource in a distributed environment, when knowledge of the location of the resource is present only at the its site of residence. Such a restriction may be enforced because the system can be made fail-soft if complicated data structures do not have to be updated when resources migrate. Broadcast searches perform an associative search in a distributed environment.

We have shown that the broadcast message may be destined for all hosts in the network, or for a subset if not all hosts support a local file system or if it is known that some of the hosts definitely do not have the file. The underlying broadcast routing capability of the communication subnet must support both types of broadcast routing. This is necessary because if there is only a broadcast-to-all capability, even if the subnet traffic is minimal, each host has to process the broadcast message. This consumes processing power unnecessarily and congests the path between the switching node and the host. Chapter 3 examines this problem in greater detail.

The broadcast search protocol must be designed such that it only locates the file. Once the file has been located the requestor

communicates with the local file system , where the file is resident, in order to make sure that the requesting process has the appropriate access rights. The system must be designed such that this communication is secure, and that no information that may compromise the requesting user process is leaked out into the distributed environment. This can be achieved by transmitting sensitive information only on a reliable point-to-point interprocess communication channel, and not by using a broadcast protocol. The communication itself can be encrypted using the NBS encryption algorithm [NBS75]. Secure communication and secure operating system design is receiving a lot of attention these days.

5.3 File Migration

One of the advantages of having a distributed file system is the ability to place files in the network so that the total cost of using them from various hosts in the network is minimized. This requirement might, in general, argue for multiple copies of some files. It is always desirable to have the processing activity take place near its program and data, in order to reduce excessive data transfers.* A system that permits file migration allows for the dynamic reconfiguration of the file locations. Files can migrate because a user process explicitly moves them from one local file system to another, or automatically. If files can migrate automatically, then an efficient DFS can be built with single copies of files. If the file system and terminal handlers are decoupled from the processor, then if the processor goes down, users can use another host and have their files migrate. Techniques for achieving this optimization are examined in this section.

5.3.1 Related Research

The problem of file allocation in a network is very similar to the plant location problem in Operations Research, and so the models used are very similar. Most of the techniques proposed so far fall into this category [Chu69, Casey72, Eswaran74, Urano74, Levin75, Chang75, Chandy76]. Techniques in estimation theory and time series analyses are used to predict what the file access patterns will be like in the near future, in order to make the optimization adaptive [Segall76].

*Cheap wideband satellite communication may change this requirement.

Chu formulates the model as an integer (0 or 1) programming problem. The model considers storage cost, transmission cost, file lengths, request and update rates of the files, the maximum storage capacity of each computer and the maximum allowable expected access time to files at each computer. The model takes into account queueing delays resulting from finite line capacity, but assumes that there are single copies of files. Casey reformulates the problem in order to make it more amenable to analysis and lets the number of copies of a file be the outcome of the optimization. An implicit assumption is that the line capacity is large enough so as not to produce queueing delays. He proposes a mechanism for obtaining the solution.

Levin and Morgan extend Casey's model by assuming a much more complicated behavior between program and data file interaction. Previous analysis assumed stationary access rates. They extend it to non-stationary but deterministic rates, and then to the case where the non-stationarity is modelled by a probability distribution.

Eswaran's result is very important. He proves that the model as formulated by Casey is polynomial complete, and so advises that heuristics should be used to find the solution. Chandy and Hewes do precisely this, and propose a near-optimal heuristic that works well.

Chang assumes that the network is hierarchically structured and proposes a model. Such a model may be appropriate for specialized data base applications, where the network has the structure similar to IBM's System Network Architecture [SNA76].

Segall proposes a complex model for estimating where a file must move to, based on how it has been moving and the behavior of request rates from various hosts.

All these models and techniques model file activity very accurately, but do not consider that the communication cost (usually delay-to-last-bit) may vary with changing load conditions, owing to the presence of other kinds of non-file traffic in the network. Many models make the implicit assumption that the line capacity is large enough so as not to produce queueing delays. In the analysis that assumes non-stationary request patterns for the files, it is very difficult to see how the data is gathered or how the results of the optimization are enforced. These models also do not assume the existence of an underlying routing algorithm that is adaptive to changing load conditions, and network failures [McQuillan74]. We believe that if automatic file migration is to occur, then distributed algorithms should be used since it is sensitive to the distributed nature of the network. Each file system maintains sufficient statistics on the usage of the files resident at its host, in order to determine where the files should move (if at all). We propose one such an algorithm in Appendix E. Its suitability is a subject for future research.

5.4 Multiple Copies of a File

If the DFS permits only single copies of files, then a lot of fail-softness of the system is lost. If multiple copies of files each having the same name are permitted, then the availability of files increases, as does the efficiency of file accesses since the various copies may be scattered around in the network. We have touched upon the problems of permitting multiple copies of files in previous sections of this chapter. We examine these problems in greater detail in this section. We assume that there is an external mechanism by which duplicate copies are created or deleted from the DFS. Automatic techniques for creating or deleting duplicate copies of files based on file usage patterns in the network is a subject for future research.

For the purpose of locating any of the duplicate copies of a file, each of them can be treated independently. We have proved (Corollary 5.1.1) that the cascade-search is unaffected by the existence of duplicate copies of read-only files. The cascade-search does not guarantee to locate the nearest copy of a file (nor all copies), since the copy located depends on the particular set of RUD pointers traversed. A broadcast search may locate the nearest copy of a file. The distributed algorithm that optimizes the location of a copy of a file (see Appendix E), does so independent of the existence of other copies. This is appropriate if the files are truly different. However, when there are duplicate copies of a file, their movement and thus the location they come to rest depends on the locations from which the requests originated. Since the cascade-search does not guarantee to

access the nearest copy, the distributed optimization may cause all the copies to bunch up, instead of being distributed through the DFS. A simple (but rather inefficient) way to overcome this problem is to have each host flush its RUD entries periodically, and then create new ones based on a pure broadcast search, thereby getting a pointer to the nearest copy.

If the multiple copies are modifiable, then the search algorithm must perform an additional set of functions besides just locating the file. We discuss these functions in the next subsection.

5.6.1 Ensuring Consistency Between Duplicate Copies

A number of problems arise if the different copies are modifiable and it is important to guarantee consistency between them, as is usually the case. This is because, owing to the inherent delays in communication and the possibility of host crashes and network failures, race conditions may cause an older copy to be accessed or leave some copies inconsistent with one another.

If all accesses were coordinated by a centralized authority, then the system could guarantee consistency between the copies since it could permit multiple readers, or only one writer at any time [Courtois71]. After a writer had finished, all the copies would have to be made consistent with one another before any other reader or writer was permitted access to the file. This may take an unpredictable amount of time if a key host goes down or the network becomes temporarily

partitioned. Such a scheme can be implemented by permitting only non-directory files to have multiple copies and centralizing the catalog. The global file space in the NSW is managed in a similar way [Schantz76, Muntz76].

We are, however, trying to get away from centralized schemes, and would like to treat all files, directory and non-directory, identically. This problem appears to be very difficult. Consistency between files can be guaranteed if some simplifying assumptions are made about the kinds of modifications permitted on the files. In a scheme proposed by Johnson and Thomas [Johnson75], the modification of an element in a data base does not depend on its old value, thereby doing away with global synchronization and locking. All modifications to a file, from all hosts are ordered so that it is possible to distinguish older modification requests (which may be delayed in the network) from more recent ones. An appropriate protocol exists to determine when a delete-entry request has been received by all data base sites, so that the entry may be garbage collected without introducing any unwanted effects. If users of a file take upon themselves the responsibility of guaranteeing consistency between copies of the file, then in many cases it is possible to define appropriate protocols to achieve this. An example of a multi-site data gathering system can be found in [Schantz74].

An algorithm for maintaining duplicate data bases has recently been proposed by Thomas [Thomas76]. This algorithm permits updates to redundantly maintained data bases from any of the data base sites.

Races between conflicting, concurrent update requests are resolved in a manner that maintains both the internal consistency and mutual consistency of the data base copies. The synchronization mechanism that resolves races does not introduce the possibility of deadlock. The data base maintenance mechanism can recover from and function effectively in the presence of communication (network) and data base site (host) failures. The algorithm is robust with respect to lost and duplicate messages, the (temporary) inability of data base managing processes to communicate with one another (due to network or host crashes), and the loss of memory (state information) by one or more data base managing processes. The algorithm does not require all the data base managing processes to be up and accessible in order that the system function correctly. The synchronization mechanisms only require pairwise interaction between the data base managing processes. The algorithm does not depend on a broadcast communication capability, but might be reformulated to take advantage of such a capability should it exist. This is an important and useful result. Such an algorithm permits the design of very general distributed file systems. Of course, restrictions on which directory or non-directory files should have multiple copies may improve the efficiency of the DFS, because schemes which guarantee consistency in such a general environment are often expensive.

5.5 Conclusions

We have shown the need for a distributed file system, and that the basic problems in its design are:

- (i) file naming conventions,
- (ii) providing simple and quick ways for searching for a file, and checking its access rights, even when it moves,
- (iii) ensuring consistency between duplicate copies of files,
- (iv) and designing techniques for permitting the automatic migration of files.

The DFS presented in this chapter uses distributed algorithms for searching for files, and for moving them in order to minimize the cost of using them by all users (Appendix E). The need for an efficient underlying broadcast routing capability in the subnet has been demonstrated, since such a capability permits the use of broadcast searches. If broadcast searches can be used, then the search algorithm for finding a file and the catalog structure can be simplified. We believe that these ideas can be extended to a file system that

- (i) has multiple copies of read-write files,
- (ii) has finite file system storage at each host,
- (iii) permits some damage to the table entries, and

(iv) has appropriate cost measures for permitting transfer of pages of an open file over an extended period of time.

The problem of guaranteeing consistency among duplicate copies of files is a very difficult problem, and solutions using distributed control are only now emerging. The suitability of the distributed optimization in various network topologies, using different routing algorithms, and under various file usage patterns, is a topic for future research.

CHAPTER 6

CONCLUSIONS AND FUTURE RESEARCH

6.1 Conclusions

This thesis has investigated the design and analysis of broadcast routing algorithms in store-and-forward packet switched computer networks. We have described a few applications for broadcast protocols in distributed computing environments. In particular, we have shown in detail how the catalog of a distributed file system could be structured in a simple way, if the system could make use of efficient reliable broadcast protocols.

We have described five alternatives to transmitting separately addressed packets from the source to the destinations. The algorithms have been compared qualitatively, in terms of memory requirements, ease of implementation, adaptiveness to changing network conditions, and reliability. The algorithms have also been compared quantitatively, in terms of the number of packet copies generated to perform broadcast, and the delays to propagate the packet to all destinations. In order to compare the algorithms in this fashion, lower bounds on the performance measures were determined for all the algorithms by examining regular graphs.

The properties of reliable broadcast protocols at the host level emerge from the reliability of the routing algorithms and the applications for the protocols. We have examined the tradeoffs between

global and subgroup broadcast routing. We believe that communication subnets should support both capabilities in the form of multi-destination addressing, and reverse path forwarding respectively.

An outcome of the investigation of broadcast routing algorithms was the formulation of two distributed (parallel) algorithms for constructing minimal spanning trees. We believe that these algorithms are the first of their kind. The formulation of such algorithms has made the problems affecting the design of distributed algorithms in network environments clearer. These minimal spanning tree algorithms can be used in broadcast routing, as well as other networks like the Packet Radio Network.

6.2 Suggestions for Future Research

We have indicated topics for future research during the course of this dissertation and briefly summarize the important ones again.

The determination of the complexity of the distributed minimal spanning tree algorithms is an interesting subject for future research. The factors that influence the complexity are the degree of parallelism, the asynchrony of internode communication, the number of signals transmitted, the length of signals, the overhead of using a broadcast routing scheme to deliver signals to nodes in the same fragment, and the data structures representing the fragment state and edge information. All these factors are not independent of each other. We believe that simulation under various conditions may be an appropriate way to determine the complexity of such algorithms. The suitability of the various information gathering schemes by the master node must also be determined. The reinitialization protocol in the adaptive algorithm was very simple. More esoteric protocols may be developed with additional properties that make the algorithm more robust. In general, the design of robust distributed algorithms is an important topic for future research, as well.

The design of reliable broadcast protocols analogous to reliable interprocess communication protocols is a very important subject for future research. The structure of such protocols is determined by the application and the reliability of the underlying broadcast routing algorithms. Efficient subgroup broadcast routing algorithms must also be designed since global broadcast routing has large overhead in large

multi-purpose communication subnets. Restricted multi-destination addressing must be investigated further, since it appears to be a good compromise for both subgroup and global broadcast routing.

The performance evaluation of the broadcast routing algorithms in the presence of background traffic, interference by packets of the same broadcast, and complex cost measures must be performed. We have not assumed that the links of the subnet can have different dollar costs, and that a user may wish to minimize not delay and number of packet copies, but the dollar cost of performing broadcast. The suitability of using regular graphs as ideal networks must be examined further, as the process of converting a given network topology into a regular graph may be a useful design heuristic.

The design of distributed file systems has many open problems. There are problems related to naming conventions, search algorithms, maintenance of consistency between duplicate copies of files, automatic creation of duplicates for efficiency and reliability purposes, and automatic file migration. The distributed file migration algorithm of Appendix E must be investigated in greater detail, to see how well it performs its optimization under different request patterns and network topologies. The algorithm for maintaining consistency between duplicate data bases [Thomas76] must be investigated for efficiency and ease of implementation. Efficient and reliable techniques for performing synchronization between processes in a distributed environment must also be developed. The design of such file systems will make it possible to reference large distributed data bases, and make it possible to design

systems that can be easily expanded. We hope to see the development of general purpose distributed operating systems.

APPENDIX A

DETAILED TABLE OF CONTENTS

<u>Subject</u>	<u>Page</u>
ABSTRACT	iv
ACKNOWLEDGEMENTS	vi
TABLE OF CONTENTS	viii
LIST OF TABLES	xi
LIST OF FIGURES	xii
CHAPTER	
1 INTRODUCTION	1
1.1 Introduction	1
1.2 Summary	7
2 DISTRIBUTED MINIMAL SPANNING TREE ALGORITHMS	9
2.1 Introduction	9
2.2 Construction Principles for Minimal Spanning Trees . .	13
2.2.1 Prim's Principles	14
2.2.2 Existing Algorithms	15
2.3 Parallel MST algorithms	17
2.3.1 Properties of Distributed MST Algorithms . . .	18
2.3.2 Transforming a Network into one with Distinct Edge Costs	19
2.4 A Static Distributed MST Algorithm	21
2.4.1 The Basic Model	21
2.4.2 Definitions	22

2.4.3	Statement of the Algorithm	28
2.4.3.1	State Information at Each Node	29
2.4.3.2	Internode Communication	30
2.4.3.3	Associated Routines	31
2.4.3.4	The Main Program	32
2.4.4	Analysis of the Algorithm	33
2.4.4.1	Termination of the Algorithm	36
2.4.4.2	Factors Influencing Complexity	37
2.4.5	Networks in which the Edge Costs are not Distinct	39
2.4.6	Conclusions	41
2.5	An Alternate Model	43
2.5.1	Ways of Gathering Information about the Fragment	47
2.6	An Adaptive Distributed MST Algorithm	50
2.6.1	The Basic Model	51
2.6.2	Statement of the Algorithm	55
2.6.2.1	State Information at Each Node	55
2.6.2.2	Internode Communication	56
2.6.2.3	Associated Routines	57
2.6.2.4	The Main Program	59
2.6.3	Termination and Reinitialization of Computation Phases	60
2.6.4	Constructing the MST from Scratch	68
2.6.5	A Very General Environment	69
2.6.6	The Packet Radio Network Environment	72
2.6.7	Analysis of the Algorithm	75
2.6.8	Conclusions	76

Detailed Table of Contents	214
2.7 Conclusions	78
3 BROADCAST ROUTING ALGORITHMS	79
3.1 Introduction	79
3.2 Separately Addressed Packets	81
3.3 Multi-Destination Addressing	84
3.4 Hot Potato Forwarding	88
3.5 Source Based Forwarding	91
3.6 Reverse Path Forwarding	93
3.6.1 The Simple Scheme	93
3.6.2 The Optimal Scheme	95
3.6.3 Reliability Issues	99
3.7 Forwarding along a Spanning Tree	102
3.8 Reliability of Broadcast Routing Protocols	105
3.9 Global vs Subgroup Broadcast Routing Algorithms	108
3.10 Conclusions	110
4 PERFORMANCE EVALUATION OF BROADCAST ROUTING ALGORITHMS	114
4.1 Introduction	114
4.2 Performance Measures	116
4.3 Regular Graphs	118
4.3.1 Some Lower Bounds in Regular Graphs	125
4.4 Separately Addressed Packets (SAP)	129
4.4.1 Number of Packets Transmitted	129
4.4.2 Broadcast Delay	129
4.5 Multi-Destination Addressing (MDA) and Source Based Forwarding (SBF)	139

Detailed Table of Contents	215
4.6 Hot Potato Forwarding	142
4.7 Reverse Path Forwarding	146
4.8 Minimal Spanning Tree Forwarding	149
4.8.1 Broadcast Delays	149
4.9 Comparison of the Different Schemes	154
4.10 Performance in the ARPANET	157
4.11 Conclusions	162
5 DISTRIBUTED FILE SYSTEMS: AN APPLICATION	163
5.1 Introduction	163
5.2 User Interface and Catalog Structure	169
5.2.1 Centralized or Duplicated Catalogs	170
5.2.2 Partitioning the Catalog Based on Pointers	173
5.2.3 Partitioning the Catalog Based on Pure Broadcast	177
5.2.4 Partitioning the Catalog Based on File Usage Patterns	182
5.2.4.1 Search Algorithms	183
5.2.4.2 Properties of the Cascade-Search Algorithm	186
5.2.4.3 Reliability of the Cascade-Search	190
5.2.5 A DFS with Constraints on Links	193
5.2.6 Broadcast Search Protocols	196
5.3 File Migration	198
5.3.1 Related Research	198
5.4 Multiple Copies of a File	201
5.6.1 Ensuring Consistency Between Duplicate Copies	202
5.5 Conclusions	205

Detailed Table of Contents	216
6 CONCLUSIONS AND FUTURE RESEARCH	207
6.1 Conclusions	207
6.2 Suggestions for Future Research	209
APPENDIX A DETAILED TABLE OF CONTENTS	212
APPENDIX B AN EXAMPLE OF REDUNDANT COMPUTATION OWING TO COMMUNICATION DELAYS	217
APPENDIX C THE ARPANET ROUTING ALGORITHM	221
APPENDIX D SUM OF DELAYS FOR A COMPLETELY FILLED PRIMARY SUBTREE	224
APPENDIX E A DISTRIBUTED FILE MIGRATION ALGORITHM	228
E.1 Introduction	228
E.2 A Distributed File Migration Algorithms	228
E.2.1 Algorithm I	232
E.2.2 Algorithm II	233
E.2.3 Algorithm III	237
E.3 Discussion	238
E.4 Finite Storage and Differing Storage Costs	238
E.5 An Observation	239
E.6 Conclusions	240
REFERENCES	242

APPENDIX B

AN EXAMPLE OF REDUNDANT COMPUTATION OWING TO COMMUNICATION DELAYS

We show how redundant computation may be performed when constructing the minimal spanning tree using the static algorithm described in section 2.4. Extra but harmless computation is performed because of delays in transmitting signals. Consider the construction of the MST in figure B.1.

Let us assume that at time T_0 the following conditions exist:

- (i) The subtree consisting of nodes H, K, I, and E has been constructed and E is active (referred to as {H,K,I,E; E active}).
- (ii) The subtree {P,O,L; L active} has been constructed.
- (iii) The subtree {A,D; D active} has been constructed.
- (iv) The subtree {C,G,F,J,M,N; F active} has been constructed.
- (v) The signal from B to D creating BD as a marked branch has not yet arrived.

Let us assume that the active nodes transmit the following signals at time T_0 :

- (1) E signals D making ED an unmarked branch and transferring master control to D.

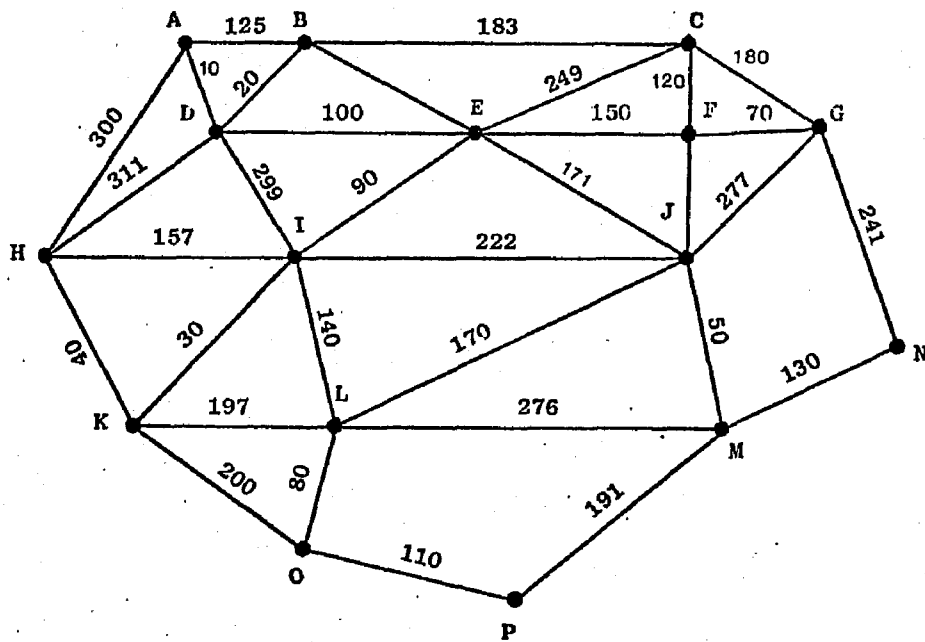


Figure B.1a A NETWORK.

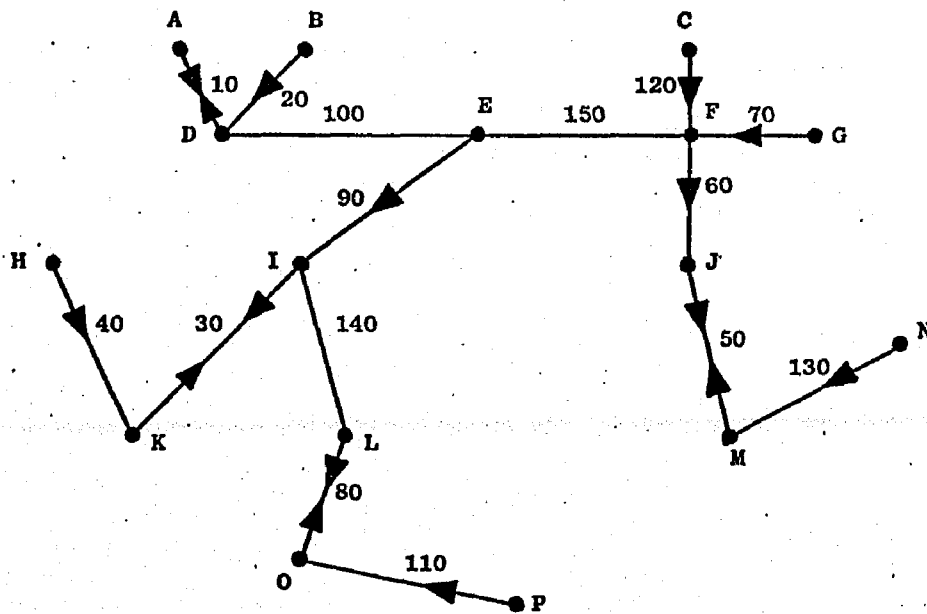


Figure B.1b THE MINIMAL SPANNING TREE FOR THE NETWORK.

(2) L signals I making LI an unmarked branch and transferring master control to I.

(3) D signals B making DB an unmarked branch and transferring master control to B.

(4) F signals E making FE an unmarked branch and transferring master control to E.

At time T_1 signals corresponding to (v), (3) and (2) arrive at their respective destinations causing the generation of the following signals:

(I) B is master and it transfers master control back to D since it thinks there is a minimum cost edge incident at D, connecting the fragment it is part of, to another fragment.

(II) I is master and it transfers master control to E.

At time T_2 the signal corresponding to (I) arrives at D, and causes the generation of the following signal:

(a) D is active and signals E making DE an unmarked branch and transferring master control to E.

At time T_3 signals corresponding to (I) and (a) reach their respective destinations, and assuming that D unambiguously becomes master it generates the following signal:

(A) D signals I to become master.

At time T_4 the signals corresponding to (II) and (A) arrive at their respective destinations and cause the generation the following signals:

(a1) E signals F making EF an unmarked branch and transferring master control to F.

(a2) I signals E to become master.

At time T_5 signals corresponding to (5) and (a1) arrive at their respective destinations, and assume that F unambiguously becomes master. F will upon examining its fragment state discover that the MST has been constructed and broadcast a 'done' signal.

At time T_6 the signal corresponding to (a2) arrives at E. E becomes master and discovers that the MST has been constructed. It too broadcasts a 'done' signal.

There are no more signals in the system except 'done' signals and the algorithm will terminate. This example shows the type of race conditions that can occur and how the algorithm copes with them.

APPENDIX C

THE ARPANET ROUTING ALGORITHM

In this appendix we describe part of the routing algorithm originally installed in the ARPANET. Much of this appendix is taken verbatim from [McQuillan74]. Details of the routing algorithms and their performance can be found in McQuillan's thesis.

The ARPANET routing algorithm can be summarized as follows. This algorithm directs each packet to its destination along a path for which the total estimated transit time is smallest. This path is not determined in advance. Instead, each IMP individually decides which line (link, edge) to use in transmitting a packet addressed to another destination. This selection is made by a simple table lookup procedure. For each possible destination, an entry in the routing table designates the appropriate next line in the path.

Each IMP maintains a NETWORK_DELAY_TABLE which gives an estimate of the delay it expects a packet to encounter in reaching every possible destination over each of its output lines. This table and other tables mentioned below are shown in figure C.1 as kept by IMP 2, for example. Thus, the delay from IMP 2 to IMP 5 using line 3 is found to be 4 in the NETWORK_DELAY_TABLE. Periodically, every $2/3$ of a second, the IMP selects the minimum delay to each destination and puts it in the MINIMUM_DELAY_TABLE. It also notes the line giving the minimum delay and keeps the number of the line in the ROUTING_TABLE for use in routing

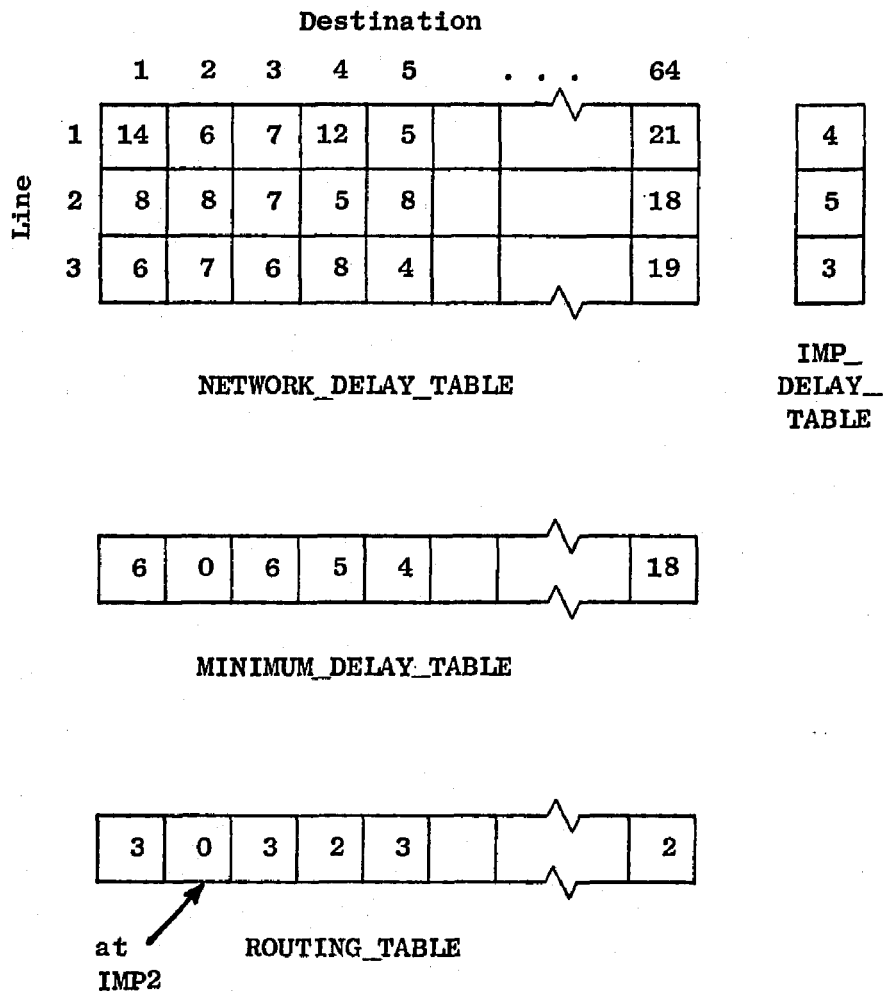


Figure C.1. THE ARPANET ROUTING TABLES IN AN IMP.

packets. Also every $2/3$ of a second, the IMP passes its MINIMUM_DELAY_TABLE to each of its immediate neighbors, that is, it sends the MINIMUM_DELAY_TABLE out each of its phone lines. Of course, before the MINIMUM_DELAY_TABLE is transmitted to the neighboring IMPs, the IMP sets the minimum delay to itself to zero.

Since all the neighbors of an IMP are also sending out their MINIMUM_DELAY_TABLE every $2/3$ second, with their own entry set to zero, an IMP receives a MINIMUM_DELAY_TABLE from each of its neighbors every $2/3$ second. These tables are read in over the rows of the NETWORK_DELAY_TABLE as they arrive. The row to be written over is the row corresponding to the phone line that the arriving MINIMUM_DELAY_TABLE came in over. After all the neighbors' estimates have arrived, the IMP adds the delay saved by the IMP itself to the neighbors' estimates. This is done by adding the IMP_DELAY_TABLE to each column of the NETWORK_DELAY_TABLE. Thus the IMP has an estimate of the total delay to each destination over the best path to that destination.

In parallel with this computation, the IMPs also compute and propagate shortest path information in a similar fashion. This information is used only in the determination of connectivity. An upper limit of the number of lines in the longest path in the network is used as the cut-off for disconnected or nonexistent nodes.

APPENDIX D

SUM OF DELAYS FOR A COMPLETELY FILLED PRIMARY SUBTREE

In this Appendix we simplify the expression for the sum of delays for a completely filled primary subtree when separately addressed packets are sent to each destination.

The sum of delays for a completely filled primary subtrees is:

$$\sum_{j=1}^m \sum_{i=LB}^{UB} (j + NPS - i).$$

where

$$LB = 1 + \sum_{k=1}^{j-1} (D-1)^{k-1}$$

and

$$UB = \sum_{k=1}^j (D-1)^{k-1}.$$

In order to simplify this expression, let

$$a = \sum_{k=1}^{j-1} (D-1)^{k-1} \quad \text{and} \quad b = \sum_{k=1}^j (D-1)^{k-1}.$$

Therefore, the inner summation becomes:

$$\sum_{i=1+a}^b (j+NPS-i)$$

$$\begin{aligned}
&= \sum_{i=0}^b (j+NPS-i) - \sum_{i=0}^a (j+NPS-i) \\
&= (b+1)*(j+NPS) - b(b+1)/2 - (a+1)*(j+NPS) + a(a+1)/2 \\
&= (j+NPS)*(b-a) + \{a(a+1) - b(b+1)\}/2 \\
&= (1/2)*\{2(j+NPS)*(b-a) + (a+b)*(a-b) + (a-b)\} \\
&= (1/2)*\{2(j+NPS)*(D-1)^{j-1} - (b-a)*(1+a+b)\} \\
&= (1/2)*[2(j+NPS)*(D-1)^{j-1} - (D-1)^{j-1}\{1+(D-1)^{j-1}-1\}/2 \\
&\quad + (D-1)^{j-1}/(D-2)\}] \\
&= (1/2)*[2(j+NPS)*(D-1)^{j-1} - \frac{(D-1)^{j-1}}{(D-2)}\{D+(D-1)^{j-1}+(D-1)^{j-4}\}].
\end{aligned}$$

Now performing the outer summation we get:

$$\begin{aligned}
&\sum_{j=1}^m (j+NPS)*(D-1)^{j-1} - \sum_{j=1}^m \frac{D(D-1)^{j-1}}{2(D-2)} - \sum_{j=1}^m \frac{(D-1)^{2(j-1)}}{2(D-2)} \\
&+ \sum_{j=1}^m \frac{(D-1)^{2j-1}}{2(D-2)} + \sum_{j=1}^m \frac{4(D-1)^{j-1}}{2(D-2)}.
\end{aligned}$$

The first, second and last terms can be computed together giving:

$$\begin{aligned}
&\sum_{j=1}^m \{NPS - D/(2(D-2)) + 4/(2(D-2))\}*(D-1)^{j-1} + \sum_{j=1}^m j*(D-1)^{j-1} \\
&= NPS*\{NPS - (D-4)/(2(D-2))\} + \sum_{j=1}^m j*(D-1)^{j-1}.
\end{aligned}$$

$$\begin{aligned}
\text{and } \sum_{j=1}^m j*(D-1)^{j-1} &= \frac{d}{dD} \sum_{j=1}^m (D-1)^j \\
&= \frac{d}{dD} [\{(D-1)^{m+1}-1\}/(D-2) - 1] \\
&= \{(m+1)*(D-1)^m*(D-2) - (D-1)^{m+1} + 1\}/(D-2)^2 \\
&= \{(m+1)*(D-1)^m*(D-2) - (D-1)^{m+1}\}/(D-2)^2 + 1/(D-2)^2 \\
&= (D-1)^m \{m(D-2)-1\}/(D-2)^2 + 1/(D-2)^2 \\
&= (D-1)^m * m*(D-2)/(D-2)^2 - (D-1)^m/(D-2)^2 + 1/(D-2)^2 \\
&= m*(D-1)^m/(D-2) - \{(D-1)^m-1\}/(D-2)^2 \\
&= m*(D-1)^m/(D-2) - NPS/(D-2).
\end{aligned}$$

Since $(D-1)^m = NPS*(D-2) + 1$, the above further simplifies to:

$$\begin{aligned}
&m*\{NPS*(D-2)+1\}/(D-2) - NPS/(D-2) \\
&= m*NPS + (m-NPS)/(D-2).
\end{aligned}$$

∴ The first, second and last terms simplify to:

$$NPS*\{NPS-(D-4)/(2(D-2))\} + m*NPS + (m-NPS)/(D-2).$$

The third and fourth terms simplify to:

$$-(1/(2(D-2))) * \sum_{j=1}^m \{(D-1)^{2j-2} + (D-1)^{2j-1}\}$$

$$\begin{aligned}
&= -\sum_{j=1}^m (D/(2(D-2))) * (D-1)^{2(j-1)} \\
&= -(D/(2(D-2))) * \{ (D-1)^{2m} - 1 \} / \{ (D-1)^2 - 1 \} \\
&= -(D/(2(D-2))) * \{ (D-1)^{m-1} * (D-1)^{m+1} \} / \{ (D-1-1) * (D-1+1) \} \\
&= -(1/(2(D-2))) * [NPS * \{ (D-1)^{m-1} + 2 \}] \\
&= -(NPS/2) * \{ (D-1)^{m-1} / (D-2) + 2 / (D-2) \} \\
&= -NPS^2/2 - NPS/(D-2).
\end{aligned}$$

Therefore the total delay now becomes:

$$\begin{aligned}
&NPS * \{ NPS - (D-4) \} / \{ 2(D-2) \} + m * NPS + (m - NPS) / (D-2) \\
&- NPS^2/2 - NPS / (D-2). \\
&= NPS^2 - (D * NPS - 4NPS) / \{ 2(D-2) \} - NPS^2/2 - NPS / (D-2) \\
&\quad + m * NPS + (m - NPS) / (D-2) \\
&= NPS^2/2 + \{ -D * NPS + 4NPS - 2NPS \} / \{ 2(D-2) \} + m * NPS + (m - NPS) / (D-2) \\
&= NPS^2/2 - NPS * (D-2) / \{ 2(D-2) \} + m * NPS + (m - NPS) / (D-2) \\
&= NPS^2/2 - NPS/2 + m * NPS + (m - NPS) / (D-2) \\
&= NPS * \{ NPS - 1 + 2m \} / 2 - (NPS - m) / (D-2).
\end{aligned}$$

APPENDIX E

A DISTRIBUTED FILE MIGRATION ALGORITHM

E.1 Introduction

In this Appendix, we describe a distributed algorithm for locating files in a computer network. This algorithm is applicable mainly in a store-and-forward network since it attempts to minimize the communication cost for accessing the files. In an inherently broadcast network the communication cost to access a file is independent of its location, as long as it is not present locally.

Therefore, we believe that the position of a file should move closer to the host that accesses it most often. Files will move around in the DFS, and may even reside at a host where they are not being referenced because hosts nearby access it. Drawing an analogy with force fields, at each requesting site the magnitude of the vector is a function of the request rate and the retrieval delay for the file. The direction of the vector is from the file to the requesting host. The file should be placed in the two dimensional space such that the sum of all the vectors is zero. Hence the file is in "equilibrium".

E.2 A Distributed File Migration Algorithm

A distributed algorithm is now proposed for performing automatic file migration. It is based on the "climb" algorithm used as a

replacement algorithm in paging systems [Coffman73], except that files can climb in more than one direction. Assume that there is only one copy of any file in the DFS, that there is infinite file storage space at each local file system, and that the cost of storage is the same at every host. We believe that files will not move about very quickly, thereby justifying the cascade-search. Of course, the interval chosen for deciding whether to move the files may affect the validity of RUD entries.

Let us assume that there is only one host connected to each switching node. Assume that each local file system has knowledge of which hosts are its physical neighbours. It includes its own host in this set. Figure E.1a shows the physical neighbor hosts for host 1. Every time a particular file is accessed, the file system on which it resides calculates the cost to transfer the file to the requestor. The cost is mainly a function of the time taken for this transaction. The file system also records via which physical neighbor the file first went. (In a store and forward network, there may be adaptive routing, and packets of the file may take different paths, and so the physical neighbor recorded is the one via which most packets went for this transaction). Hence, information on the cumulative-cost for a given direction and file is maintained at each local file system. This information is used to determine which files should move and to which physical neighbor. Hence each file climbs from its present place to another until it reaches what it thinks is the appropriate local file system. Each local file system in the DFS may attempt to transfer a

file only to one of its immediate physical neighbors. Hence, although requests for a file are coming from all hosts in the network, for purposes of the optimization it appears as though the requests are concentrated at the neighboring hosts.

Any dynamic algorithm may lead to instabilities of the kind where the file keeps moving around without any apparent benefit. This could happen if a file kept oscillating between two nodes, or if the movement of one file caused many others to move unnecessarily. The latter situation could arise if the local file systems had finite storage. Therefore a file should be moved to a neighbor file system only if the savings in cost in doing so is larger than a threshold - at least the cost of moving it from one host to another.

Different strategies for making the decision to cause a file to migrate will now be considered. First some definitions are introduced.

Let U be the Utilization matrix, where an element $u(i,n)$ is the utilization of the file i from neighbor n . U is of dimension $NFN \times NPN$ where NFN is the Number of Files at this Node, and NPN is the Number of Physical Neighbors of this node. (The node counts itself as a physical neighbor). An element of U corresponds to the cumulative-cost for a given direction and file. Its value is dependent on the time interval over which the measurement was made, and may also be determined using a predictive algorithm on the measurements. Figure E.1b illustrates what U may look like at a host.

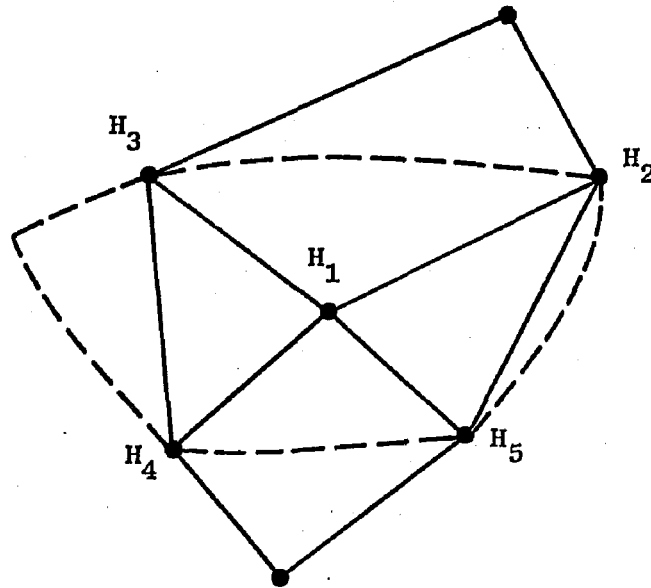


Figure E.1a. PHYSICAL NEIGHBOR HOSTS OF HOST 1.

		Hosts				
		1	2	3	4	5
Files	1	0	0	0	0	0
	2	3	2	4	8	1
	3	32	7	8	5	20
	4	2	15	22	8	7
	5	16	0	0	54	12
	6	5	1	8	6	0

Figure E.1b. THE UTILIZATION MATRIX AT HOST 1. (Units are arbitrary.)

Let S be the Selection matrix, where an element $s(i,n)$ is the savings in cost made by transferring file i to neighbor n . S is of dimension $NFN \times NPN$. The n that gives the maximum saving for a particular file is the appropriate neighbor to transfer the file.

Let D be the threshold matrix, where an element $d(i,n)$ is the expected cost for transferring file i to neighbor n . D is of dimension $NFN \times NPN$.

The elements of S are determined from the measurements on file utilization over a period of time. At the end of this interval, if there are any files, whose migration would reduce the cost of their usage, then they are transferred to the neighbor that results in maximum cost reduction. The elements of S are ordered by the saving in cost achievable, if the file was to be moved to a neighbor. The file system should attempt to move those files that would result in maximum saving first.

We now show different ways of finding S .

E.2.1 Algorithm I

The simplest algorithm is one in which

$$S = U - D. \quad (E.1)$$

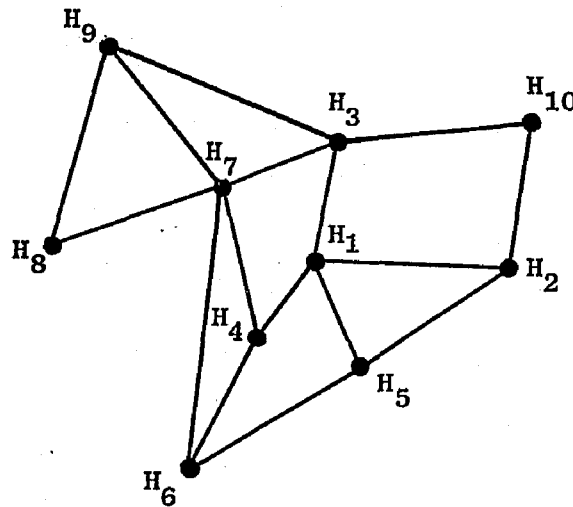
In other words, a file is transferred to the neighbor that makes greatest utilization of it. The algorithm makes no use of the topological relationship of the host on which it is executing with the

rest in the network. Hence it is possible that transfer of a file in the direction of greatest utilization may not contribute to lowering the overall cost of usage of the file. Figure E.2 illustrates an example in which the decision based on this simple algorithm is sub-optimal.

E.2.2 Algorithm II

This algorithm makes some simplifying assumptions regarding the topology and routing algorithms of the network, thereby attempting to overcome the sub-optimality of Algorithm I. It assumes that when a file is moved to another neighbor, then the requests and transfer of data from the other neighbors will include, in their path, the switching node to which the host, on which the file originally resided, is connected. In figure E.3, if the file was originally at host 1, and is now moved to 2, then traffic from hosts 3, 4, 5 will pass through node 1. In deciding which host (if any) a file should move to, the local file system at host 1 decides if there is any saving in the total cost of using the file after it has moved. If the file was moved to host 2, then the cost of usage from hosts 1, 3, 4 and 5 would increase, while that from 2 would decrease. The threshold cost and the cost of transfer over the various links have to be taken into account as well.

We now show how elements of S are calculated. Let $L(g)$ be the cost of communication from the file system at the host on which this algorithm is executing to its physical neighbors g , for all $1 \leq g \leq NPN$. The host on which the algorithm executes is $g=1$. These communication costs are illustrated in figure E.3. The savings in cost in using a



Assume that a file F is at Host 1 and that the appropriate row of U there is

	Host	1	2	3	4	5
File		15	30	25	25	2
F						
⋮						

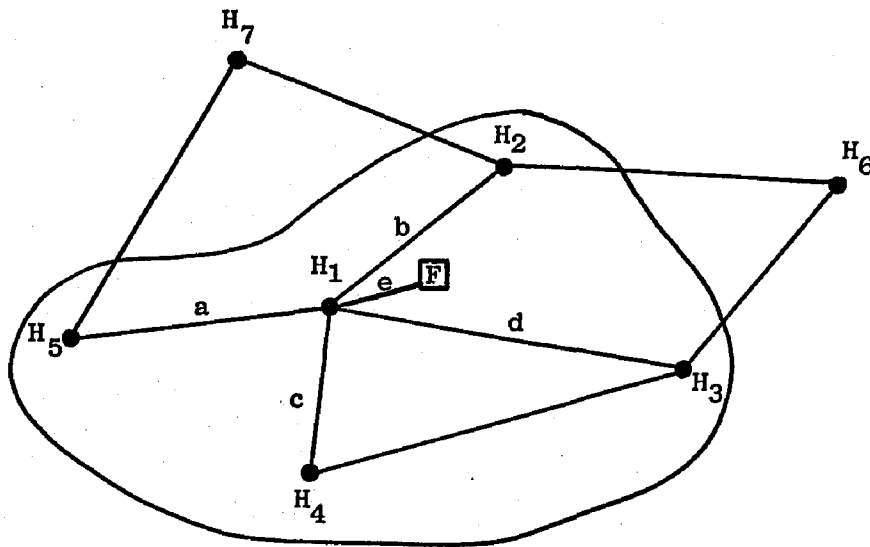
The decision to transfer the file to Host 2 would be sub-optimal if the distribution of actual requests for F were

	Host	1	2	3	4	5	6	7	8	9	10
File		30	30	1	5	2	0	10	0	12	0
F											
⋮											

Assuming infinite line capacity and minimum distance (hop) routing, the cost (request rate \times distance*) of keeping the file at 1 is $(15 + 30 + 25 + 25 + 2) = 97$ units while at host 2 it would be $(30 + 15 + 2 + 10 + 2 + 30 + 36) = 125$ units.

* Assume the distance for satisfying requests to locally resident files is 1/2 unit while link costs are equal to 1 unit.

Figure E.2. SUB-OPTIMAL DECISION USING ALGORITHM I.



a, b, c, d, e are link costs.

Communication costs L as seen by host 1 are:

$$L(1) = e; \quad L(2) = b + e; \quad L(3) = d + e;$$

$$L(4) = c + e; \quad L(5) = a + e$$

Figure E.3. HOST 1 AND ITS PHYSICAL NEIGHBORS.

file after transferring it to neighbor n , is equal to the difference between the cost of keeping the file at the current host and at neighbor n . The cost of using a file i at its current residence is equal to the sum of its utilizations from different neighbors, and is equal to

$$\sum_{g=1}^{NPN} u(i,g). \quad (E.2)$$

Now, if the file were to move to neighbor n , the cost of using it at n would be different. In figure E.3, if the file were to move from host 1 to 2, then the utilization from host 5 would increase to $(L(5)-L(1)+L(2))/L(5)$ of its original value. This is under the assumption that traffic passes through the switching node to which host 1 is connected, and that $L(1)$ relative to all hosts is the same i.e. the communication cost of retrieving a file from local file storage is the same for all hosts. Generalizing, we conclude that when the file moves to neighbor n , the utilization from all neighbors g , $g \neq n$, increases to $(L(g)-L(1)+L(n))/L(g)$ times its original value. The utilization from n itself increases to $L(1)/L(n)$ times its original value. Therefore the cost of using a file i at a neighbor n is equal to

$$u(i,n)*L(1)/L(n) + \sum_{\substack{g=1 \\ g \neq n}}^{NPN} u(i,g)*(L(g)-L(1)+L(n))/L(g). \quad (E.3)$$

The savings in cost of using the file i when it has moved to n is given by equation (E.2) minus equation (E.3). Simplifying, elements of S are given by:

$$\begin{aligned}
 s(i,n) &= u(i,n) * (L(n) - L(1)) / L(n) - \sum_{\substack{g=1 \\ g \neq n}}^{NPN} u(i,g) * (L(n) - L(1)) / L(g) - d(i,n) && ;n=2, \dots, NPN \\
 s(i,n) &= 0 && ;n=1
 \end{aligned}
 \tag{E.4}$$

The savings are relative to the cost of keeping the file at this host.

E.2.3 Algorithm III

This algorithm is an extension of the previous one, and attempts to take into account in greater detail the path taken by file traffic from the neighbors once the file has been moved. If the traffic does not pass through the node corresponding to the original host (as in Algorithm II), then it must pass through some other node such that the cost is less (by virtue of the routing algorithms of the communication subnet). These alternate routes depend on the topology and dynamically changing load conditions. These factors could be taken into account in the following way:

$$\begin{aligned}
 s(i,n) &= u(i,n) * (L(n) - L(1)) / L(n) - \sum_{\substack{g=1 \\ g \neq n}}^{NPN} r(g,n) * u(i,g) * (L(n) - L(1)) / L(g) \\
 &\quad - d(i,n) && ;n=2, \dots, NPN \\
 s(i,n) &= 0 && ;n=1
 \end{aligned}
 \tag{E.5}$$

where $r(g,\bar{n})$ is the topology and routing factor for each neighbor pair. This factor is always less than or equal to one. R is the topology and routing matrix. It is of dimension $NPN \times NPN$. Further research is

required to determine how these factors are calculated, i.e. should they be calculated dynamically or can they be assumed to be heuristic constants?

E.3 Discussion

It is hoped that this distributed optimization will approximate the same optimum as would be predicted by an algorithm that has complete knowledge of the topology and request rates. This is greatly dependent on the rate of variation of the request rates and the speed with which such an algorithm can track the changing optimum. Further, the topology and routing algorithms of the communication network may be such that the stepwise optimization causes a file to get stuck at a local minimum. The effect of network topologies and routing algorithms on the success of this optimization technique is a topic for future research.

E.4 Finite Storage and Differing Storage Costs

Section E.2 indicated that when the algorithm for placing the files in the network is as distributed as the step-wise optimization proposed, the cost for utilizing the file has to be based on the time to successfully transfer the file, the rate of usage from each host, and the size of the file. This is because these are the only parameters that the file system on which the file resides can measure.

Local file systems in reality will not have infinite file storage, and so this fact must be incorporated into the distributed algorithm, if possible. We describe a possible technique. When the file system

storage is nearly completely used (say 90%), then the file system must try to archive those files that were not accessed (for say 30 days). If all the files were accessed, and there are no candidates for migration, then the file system must do something, otherwise it can not partake in the distributed optimization. This might be one such solution to the finite storage problem, but results in a breakdown of the optimization process once any host has made the decision not to accept files from its neighbors. Alternatively, the host can move a file such that its movement will cause the least loss in optimization. Hopefully such a strategy will cause files that are not very often accessed to find their "correct" location. These ideas are sheer speculation since their effect on the optimization process has not been determined.

We also feel that the distributed optimization proposed can not, in its present form, be made to incorporate differing cost of storage at each host. Even if a host knew the cost of storage of its physical neighbors, this knowledge is not useful since if it were incorporated in the cost function, it might result in a file getting stuck at a local minima.

E.5 An Observation

Traditionally, protocols have been structured in a layered fashion [Crocker72]. This is very appropriate when interfaces between various levels are clean, and state information at lower levels is not required at higher levels. However, a distributed operating system that is trying to optimize the location of its resources in the network must

have some knowledge of the topology of the communication subnet. The network of hosts cannot look like a fully connected network for all purposes. The amount of topological information required should be the minimal amount that guarantees some acceptable level of optimization. The algorithms presented in this section suggest that near neighbor information may be sufficient. This is, in addition, helpful when the network is expanded by adding more hosts, since the data structures of only the near neighbors of the new host have to be modified. This means that the host/switching-node interface has to be sensitive to this. The switching node will have to tell the host along which path the packets (say) were sent. Figure E.4 illustrates this interaction between the IMP and a host in the ARPANET.

E.6 Conclusions

This appendix has proposed a distributed file migration algorithm. Its suitability has yet to be determined. The algorithm must be extended to an environment where there is more than one host connected to each switching node. Cheap wideband communication may, however, make automatic file migration unnecessary.

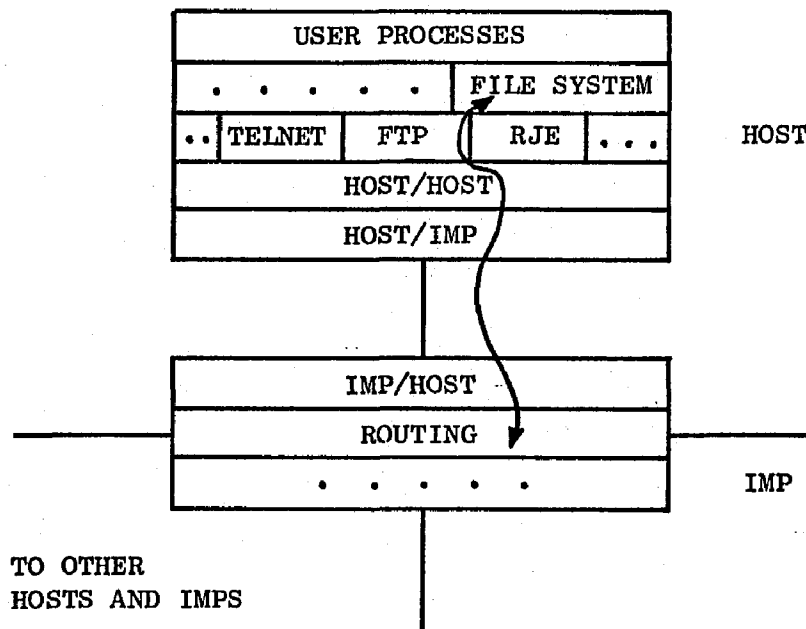


Figure E.4. PROTOCOL LAYERING IN THE ARPANET. Topology information must traverse indicated path.

REFERENCES

References are cited by the principal author's last name followed by the year of publication (and a letter to distinguish multiple publications from the same year).

INWG Notes are available from Technical Committee 6.1 of IFIP for the cost of reproduction: IFIP WG6.1, Derek L. A. Barber, Chairman, European Informatics Network, Queens Road, Teddington, Middlesex, England.

RFC's (Request for Comments) and NIC documents are available from the Network Information Center in limited numbers: Elizabeth (Jake) Feinler, Stanford Research Institute, Network Information Center, 333 Ravenswood Av., Menlo Park, Ca. 94205.

- [Abramson70] N. Abramson, "THE ALOHA SYSTEM - Another alternative for Computer Communication," Fall Joint Computer Conf., 1970, AFIPS Press, pp. 281-285.
- [Abramson73] N. Abramson, "Packet Switching with Satellites," National Computer Conf., 1973, AFIPS Press, pp. 695-702.
- [Baran64] P. Baran, S. Boehm, P. Smith, "On Distributed Communication," Rand Technical Report, Vol. I-XI, 1964.
- [Bensoussan72] A. Bensoussan, C. T. Clingen and R. C. Daley, "The Multics Virtual Memory: Concept and Design," Comm. ACM, Vol. 15, No. 5, May 1972, pp. 308-318.
- [Bentley75] J. L. Bentley and J. H. Friedman, "Fast Algorithms for Constructing Minimal Spanning Trees in Coordinate Spaces," SLAC Technical Report SLAC PUB-1665, Stanford University, December 1975.
- [Bobrow72] D. G. Bobrow, J. D. Burchfiel, D. L. Murphy, and R. S. Tomlinson, "TENEX, a paged timesharing system for the PDP-10," Comm. ACM, Vol. 15, No. 3, March 1972, pp. 135-143.
- [Booth72] G. M. Booth, "The use of distributed data bases in Information Networks," Proc. First International Conference on Computer Communication, Oct. 1972, pp. 371-375.
- [Carr70] S. Carr, S. Crocker, and V. Cerf, "Host/Host Protocol in the ARPA Network," Proc. Spring Joint Computer Conf., 1970, AFIPS Press, pp. 589-597.
- [Casey72] R. G. Casey, "Allocation of copies of a file in an information network," Proc. Spring Joint Computer Conf., 1972, AFIPS Press, pp. 617-625.
- [Cerf73] V. G. Cerf, D. D. Cowan, R. C. Mullin, R. G. Stanton, "Computer Networks and Generalized Moore Graphs," Congressus Numerantium 9, Proc. Third Manitoba Conference on Numerical Mathematics, 1973, pp. 379-398.
- [Cerf74] V. G. Cerf and R. E. Kahn, "A Protocol for Packet Network Intercommunication," IEEE Trans. on Communications, COM-22, May 1974, pp. 637-648.
- [Cerf74a] V. Cerf, Y. Dalal and C. Sunshine, "Specification of Internet Transmission Control Program," INWG Note #72, revised Dec. 1974.
- [Cerf74b] V. G. Cerf, D. D. Cowan, R. C. Mullin, R. G. Stanton, "A Lower Bound on the Average Shortest Path Length in Regular Graphs," Networks, Vol. 4, 1974, pp. 335-342.

- [Cerf74c] V. G. Cerf, D. D. Cowan, R. C. Mullin, R. G. Stanton, "Topological Design Considerations in Computer Communication Networks," in Computer Communication Networks, ed. R. L. Grimsdale and F. F. Kuo, Nato Advanced Study Institute Series, April 1974.
- [Cerf74d] V. G. Cerf, D. D. Cowan, R. C. Mullin, R. G. Stanton, "A Partial Census of Trivalent Generalized Moore graphs," Proc. Third Combinatorial Conference, Brisbane, 1974. Lecture Notes in Mathematics 452 - Combinatorial Mathematics III, Springer-Verlag.
- [Cerf74e] V. G. Cerf, D. D. Cowan, R. C. Mullin, R. G. Stanton, "Trivalent Generalized Moore Networks on Sixteen Nodes," Utilitas Mathematica, Vol. 6, 1974, pp. 259-283.
- [Cerf76] V. G. Cerf, "Obtaining Broadcast Communication from Non-Broadcast Transmission Media," private communication, Sept. 1976.
- [Chandy76] K. M. Chandy and J. E. Hewes, "File allocation in Distributed Systems," Proc. International Symposium on Computer Performance Modelling, Measurement and Evaluation, March 1976, pp. 10-13.
- [Chang75] S. K. Chang, "Data Base Decomposition in a Hierarchical Computer System," Research Report RC 5345, IBM Watson Research Center, Yorktown Heights, New York, March 1975.
- [Chu69] W. W. Chu, "Optimal File Allocation in a Multi-Computer Information System," IEEE Trans. on Computers, Vol. C-18, No. 10, Oct. 1969, pp. 885-889.
- [Coffman73] E. G. Coffman and P. J. Denning, "Operating Systems Theory", Prentice-Hall, 1973.
- [COMPASS76] Semi-Annual Technical Report, Massachusetts Computer Associates Inc. Document No. CADD-7603-0411, March 4, 1976.
- [Cosell75] B. P. Cosell, P. R. Johnson, J. H. Malman, R. E. Schantz, J. Sussman, R. H. Thomas, and D. C. Walden, "An Operational System for Computer Resource Sharing," Proc. Fifth Symposium on Operating Systems Principles, November 1975, (available as ACM Operating Systems Review, Vol. 9, No. 5, pp. 75-81).
- [Courtois71] P. J. Courtois, F. Heymans, and D. L. Parnas, "Concurrent Control with "Readers" and "Writers"", Comm. ACM, Vol. 14, No. 10, Oct. 1971, pp. 667-668.
- [Crocker72] S. Crocker, J. Heafner, R. Metcalfe and J. Postel, "Function Oriented Protocols for the ARPA network," Proc. Spring Joint Computer Conf., 1972, AFIPS Press, pp. 271-280.

- [Crocker75] S. D. Crocker, "The National Software Works: A New Method for Providing Software Development Tools using the Arpanet," Consiglio Nazionale delle Ricerche ISTITUTO DI ELABORAZIONE DELLA INFORMAZIONE meeting on 20 years of Computer Science, Pisa, June 1975.
- [Dalal74] Y. K. Dalal, "More on Selecting Sequence Numbers," Proc. ACM SIGCOMM/SIGOPS Interprocess Communication Workshop, Santa Monica, March 1975, pp. 25-36. (ACM Operating Systems Review, Vol. 9, No. 3, July 1975). Also INWG Protocol Note #4, October 1974.
- [Dalal75] Y. K. Dalal, "Establishing a Connection," INWG Protocol Note #14, March 1975.
- [Dalal76] Y. K. Dalal, "A Distributed Algorithm for Constructing Minimal Spanning Trees in Computer-Communication Networks," Technical Report No. 111, Digital Systems Lab., Stanford University, June 1976.
- [Daley65] R. C. Daley and P. G. Neumann, "A general-Purpose file system for secondary storage," Proc. Fall Joint Computer Conf., 1965, AFIPS Press, pp.213-228.
- [Daley68] R. C. Daley and J. B. Dennis, "Virtual Memory, Processes, and Sharing in MULTICS," Comm. ACM, Vol. 11, No. 5, May 1968, pp. 306-312.
- [Erdős73] P. Erdős and A. Rényi, "On the Evolution of Random Graphs," Paul Erdos: The Art of Counting - Selected Writings, edited by J. Spencer, The MIT Press, 1973, pp. 574-617.
- [Erdős73a] P. Erdős and P. Kelly, "The Minimal Regular Graph Containing a Given Graph," Paul Erdos: The Art of Counting - Selected Writings, edited by J. Spencer, The MIT Press, 1973, pp. 354-355.
- [Eswaran74] K. P. Eswaran, "Placement of records in a file and file allocation in a Computer Network," IFIP-74, pp. 304-307.
- [Farber72] D. J. Farber and K. C. Larson, "The Structure of a Distributed Computing System - The Communication System," Proc. of the Symposium on Computer-Communications Networks and Traffic, Polytechnic Institute of Brooklyn, April 1972, pp. 21-27.
- [Farber72a] D. J. Farber and K. C. Larson, "The Structure of a Distributed Computing System - Software," Proc. of the Symposium on Computer-Communications Networks and Traffic, Polytechnic Institute of Brooklyn, April 1972, pp. 539-545.
- [Farber72b] D. J. Farber and F. R. Heinrich, "The Structure of a Distributed Computer System - The distributed File System," Proc. of ICCS 1972, pp. 364-370.

- [Frank75] H. Frank, I. Gitman and R. VanSlyke, "Packet Radio Network Design - System Considerations," National Computer Conf., 1975, AFIPS Press, pp. 217-231.
- [Fultz72] G. L. Fultz, "Adaptive Routing Techniques for Message Switching Computer Communication Networks," UCLA-ENG-7252, July 1972. (Ph.D. Thesis, UCLA).
- [Gerla73] M. Gerla, "The Design of Store-and Forward (S/F) Networks for Computer Communication," UCLA-ENG-7319, 1973. (Ph.D. Thesis, UCLA).
- [Harary69] F. Harary, Graph Theory, Addison-Wesley Publishing Co., 1969.
- [Hoffman60] A. J. Hoffman, R. R. Singleton, "On Moore Graphs with Diameters 2 and 3," IBM Journal, November 1960, pp. 497-504.
- [Johnson75] P. R. Johnson, and R. H. Thomas, "The Maintenance of Duplicate Databases," RFC #677, NIC #31507, Jan. 1975.
- [Kahn75] R. E. Kahn, "The Organization of Computer Resources Into a Packet Radio Network," National Computer Conf., May 1975, AFIPS Press, pp. 177-186.
- [Kahn76] R. E. Kahn, "Techniques for Handling Stream Traffic Via Packet Switching," Oral Presentation only, IEEE Spring Compcon, February 1976.
- [Kamoun76] F. Kamoun, "Design Considerations for Large Computer Communication Networks," UCLA-ENG-7642, April 1976. (Ph.D. Thesis, UCLA).
- [Kershenbaum74] A. Kershenbaum, "Computing Capacitated Minimal Spanning Trees Efficiently," Networks, Vol. 4, No. 4, 1974, pp. 299-310.
- [Kruskal56] J. B. Kruskal Jr., "On the Shortest Spanning Subtree of a graph and the Traveling Salesman Problem," Proc. Amer. Math. Soc., 7, 1956, pp. 48-50.
- [Lam74] S. S. Lam, "Packet Switching in a Multi-Access Broadcast Channel with Application to Satellite Communication in a Computer Network," UCLA-ENG-7429, April 1974. (Ph.D. Thesis, UCLA).
- [Levin75] K. D. Levin and H. L. Morgan, "Optimizing distributed data bases - A framework for research," Proc. NCC, 1975, AFIPS Press, pp. 473-478.
- [Madnick69] S. E. Madnick and J. W. Alsop II, "A modular approach to file system design," Proc. Spring Joint Computer Conf., 1969, AFIPS Press, pp. 1-13.

- [McKenzie72] A. A. McKenzie, "HOST/HOST Protocol for the ARPA Network," NIC #8246, January 1972.
- [McQuillan72] J. M. McQuillan, W. R. Crowther, B. P. Cosell, D. Walden and F. E. Heart, "Improvements in the Design and Performance of the ARPA Network," Proc. Fall Joint Computer Conf., 1972, AFIPS Press, pp. 741-754.
- [McQuillan74] J. M. McQuillan, "Adaptive Routing Algorithms for Distributed Computer Networks," BBN Report No. 2831, May 1974. (Ph.D. Thesis, Harvard University).
- [Metcalf72] R. M. Metcalfe, "Strategies for Interprocess Communication in a Distributed Computing System," Proc. of the Symposium on Computer-Communications Networks and Teletraffic, Microwave Research Institute of Polytechnic Institute of Brooklyn, April 1972, pp. 519-526.
- [Metcalf73] R. M. Metcalfe, "Packet Communication," Project MAC Report MAC TR-114, December 1973. (Ph.D. Thesis, Harvard University).
- [Metcalf76] R. M. Metcalfe and D. R. Boggs, "Ethernet: Distributed Packet Switching for Local Computer Networks," Comm. ACM, July 1976, Vol. 19, No. 7, pp. 395-404.
- [Muntz76] C. A. Muntz, and P. M. Cashman, "File Package: The File Handling Facility for the National Software Works (Preliminary)," Massachusetts Computer Associates Inc. Document No. CADD-7602-2011, February 20, 1976.
- [Murphy72] D. L. Murphy, "Storage organization and management in TENEX," Proc. Fall Joint Computer Conf., 1972, AFIPS Press, pp. 23-31.
- [NBS75] "Federal Information Processing Encryption Standard - Proposed Standard," Federal Register, Vol. 40, No. 149 - Friday, August 1, 1975, pp. 32395.
- [Opderbeck76] H. Opderbeck and R. B. Hovey, "Telenet--Network Features and Interface Protocols," Proc. NTG-Conference on Data Networks, Baden-Baden, West Germany, February 1976.
- [Organick72] E. I. Organick, "The Multics System: An Examination of Its Structure," The MIT Press, 1972.
- [Ornstein75] S. M. Ornstein, W. R. Crowther, M. F. Krayley, R. D. Bressler, A. Michel, F. E. Heart, "Pluribus - A reliable multiprocessor," National Computer Conf., May 1975, AFIPS Press, pp. 551-559.

- [Paoletti73] L. M. Paoletti, "AUTODIN," in R. L. Grimsdale and F. F. Kuo, editor, Computer Communication Networks, Proceedings of the NATO Advanced Study Institute on Computer Communications Networks, Sussex, U.K., Sept. 1973. Noordoff Int. Publishing, Leyden, 1975.
- [Pierce75] A. R. Pierce, "Bibliography on Algorithms for Shortest Path, Shortest Spanning Tree, and Related Circuit Routing Problems (1956-1974)," Networks, Vol. 5, 1975, pp. 129-149.
- [Pouzin73] L. Pouzin, "Presentation and Major Design Aspects of the CYCLADES Computer Network," Proc. Third Data Communications Symp., Tampa, Nov. 1973, pp. 80-85. (also NIC #18256, INWG Note #36, and SCH 511).
- [Prim57] R. C. Prim, "Shortest Connection Networks and Some Generalizations," Bell Systems Tech. J., November 1957, pp. 1389-1401.
- [Ritchie74] D. M. Ritchie, K. Thompson, "The UNIX Time-Sharing System," Comm. ACM, July 1974, Vol. 17, No. 7, pp. 365-375.
- [Roberts70] L. G. Roberts and B. D. Wessler, "Computer Network Development to Achieve Resource Sharing," Proc. Spring Joint Computer Conf., 1970, AFIPS Press, pp. 543-549.
- [Roberts72] L. G. Roberts and B. D. Wessler, "The ARPA Computer Network," in Computer Communication Networks, Abramson and Kuo, editors, Prentice Hall, 1972.
- [Rowe75] L. A. Rowe, "The Distributed Computing Operating System," Technical Report #66, Dept. of Information and Computer Science, University of California at Irvine, June 1975.
- [Rybczynski76] A. Rybczynski, B. Wessler, R. Despres, and J. Wedlake, "A new communication protocol for accessing data networks - The international packet-mode interface," Proc. National Computer Conf., 1976, AFIPS Press, pp. 477-500.
- [Schantz74] R. Schantz, "A Multi-Site Data Collection Facility", RFC #672, NIC# 31440, Dec. 1974.
- [Schantz76] R. E. Schantz, and R. E. Millstein, "The Foreman: Providing the Program Execution Environment for the National Software Works (Preliminary)," Bolt Beranek and Newman Inc. Report No. 3266, Massachusetts Computer Associates Inc. Document No. CADD-7604-0111, March 31, 1976, pp. 17-34.
- [Segall76] A. Segall, "Dynamic File Assignment in a Computer Network," IEEE Trans. on Automatic Control, Vol. AC-21, No. 2, April 1976, pp. 161-173.

- [SNA76] I.B.M. Systems Journal, Vol. 15, No. 1, 1976, pp. 4-80.
- [Stritter76] E. Stritter, "File Migration," Ph.D. Thesis, Stanford University, Dec. 1976.
- [Sunshine75] C. A. Sunshine, "Interprocess Communication Protocols for Computer Networks," Dec. 1975, DSL Technical Report #105. (Ph.D. Thesis, Stanford University).
- [TENEX-4] TENEX-4 File System, 14 Jan 1970.
- [Tobagi74] F. A. Tobagi, "Random Access Techniques for Data Transmission over Packet Switched Radio Networks," UCLA-ENG-7499, Dec. 1974. (Ph.D. Thesis, UCLA).
- [Tomlinson74] R. S. Tomlinson, "Selecting Sequence Numbers," Proc. ACM SIGCOMM/SIGOPS Interprocess Communications Workshop, Santa Monica, March 1975, pp. 11-23. (ACM Operating Systems Review, Vol. 9, No. 3, July 1975) Also INWG Protocol Note #2, August 1974.
- [Thomas73] R. H. Thomas, "A Resource Sharing Executive for the Arpanet," Proc. NCC, 1973, AFIPS Press, pp. 155-163.
- [Thomas76] R. H. Thomas, "A Solution to the Update Problem for Multiple Copy Data Bases which uses Distributed Control", BBN Report No. 3340, July 1976.
- [Tymes71] L. R. Tymes and T. W. Hall, "TYMNET--A Terminal Oriented Communication Network," Proc. Spring Joint Computer Conf., 1971, AFIPS Press, pp. 211-216.
- [Urano74] Y. Urano, K. Ono and S. Inoue, "Optimal Design of Distributed Networks," Proc. ICCG, August 1974, pp. 413-420.
- [Walden72] D. C. Walden, "A System for Interprocess Communication in a Resource Sharing Computer Network," Comm. ACM, Vol. 15, No. 4, April 1972, pp. 221-230.