1990

Designing Efficient Algorithms for Distributed Systems.

Mohan B. Sharma
Louisiana State University and Agricultural & Mechanical College

INFORMATION TO USERS

The most advanced technology has been used to photograph and reproduce this manuscript from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

U·M·I

University Microfilms International
A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
313/761-4700 800/521-0600

Order Number 9112270

Designing efficient algorithms for distributed systems

Sharma, Mohan B., Ph.D.

The Louisiana State University and Agricultural and Mechanical Col., 1990

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

DESIGNING EFFICIENT ALGORITHMS

FOR DISTRIBUTED SYSTEMS

A Dissertation

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

in

The Department of Computer Science

by

Mohan Sharma

B.S., University of Mysore, India, 1981
M.S., University of Mysore, India, 1984

August, 1990

Acknowledgements

I express my sincere gratitude and appreciation to Professor S. Sitharama Iyengar, for the advice and assistance that he provided during the course of this work. His motivation, encouragement, and guidance led me through the years.

I would like to thank my committee members: Professor Subhash Kak, Professor Donald Kraft, Professor Bush Jones, and Professor Doris Carver, for all their encouragement and support. I was fortunate to have been able to associate myself with other faculty members in my research. My association with Professor Jianhua Chen was extremely fruitful and our discussions helped me to get an insight into the problems that we discussed. I thank her for her patience, involvement, and help in my research. I am grateful to Professor Bela Bollobas for finding time to sit with me and helping me on a proof for our network recognition algorithm.

I wish to express my gratitude to Dr. B.S. Adiga who introduced me to research while I was in the Masters program. I am grateful to my dear friend and mentor Dr. Narasimha for all that he has done to me.

I have been motivated and helped by many people around me. I would like to thank Pramod, Raj Reddy, Gulati, Dr. Muruga (Sridhar), Patil, Wu Wang, Sujan, Sethu, Daryl, KK, Raj, Venkat (my partner at the pool table), Puppala, Hegde, Pathak, Raghu, and all other friends here.

My stay in Baton Rouge was made pleasant and homely by several families here. I thank Vani and family for their friendly help in the times of my difficult days. I thank Rama and family ; and Shubha and family for all their support and friendship.

Special thanks go to my mother, father, brothers, and sisters whose constant support and love that kept me at comfort here. Without their generosity, love, patience, and sacrifice this work would have been impossible. I am always grateful to them.

Table of Contents

ABSTRACT

Search for efficient algorithms for distributed systems has become an important area of computer science. This research is driven by the need to efficiently process and communicate information generated by the system. In distributed systems, topological information plays an important role in the design of fast algorithms for problems such as routing, broadcasting, and sorting. The central focus of this dissertation is the design and analysis of distributed algorithms for determining topological information in asynchronous communication networks. Specifically, we present distributed algorithms for two generic problems: distributed graph problems and network traversal problems.

Network location and network recognition are two important graph problems in distributed systems. We present unified algorithms for locating centers and medians of asynchronous communication networks. Also, we present both the centralized and decentralized versions of the algorithm. Furthermore, this is the first decentralized algorithm reported in the literature. These results are further extended to weighted networks. In addition, the unified algorithm can also be used to determine other topological parameters such as the diameter, and centroids of distributed networks.

Efficient algorithms for problems such as finding shortest paths, centers, and sorting could be designed if the network topology is known a priori. Towards this end, we solve an open problem of recognizing mesh (grid) structures. We formulate both centralized and decentralized algorithms for recognizing mesh networks. The time and message complexities of the algorithm are $O(n^{1.6})$ and O(e+nlogn), respectively, where n is the number of nodes and e is the number of edges of the graph

underlying the network.

Network traversal is a fundamental activity in a distributed system and it has been widely studied in the literature. We present efficient distributed algorithms for depth first traversal of an asynchronous communication network and show the usefulness of this algorithm in deriving efficient solutions to the problems related to network learning.

Finally, we discuss application of some of these algorithms in distributed sensor networks.

CHAPTER 1

INTRODUCTION

1.1. Overview

New parallel hardware architectures, interconnection schemes, and algorithm paradigms have evolved from the recent developments in parallel processing technologies. The sheer diversity of this technological innovation has reduced drastically, the cost of hardware processing units. This has resulted in the development of computer networks and interconnection of various computers which have become viable and cost-effective. Besides being reliable, computer networks offer several several advantages to the users including interactive computing, resource sharing, and cooperation. Compared to the traditional 'monolithic' systems (those controlled by a centralized operating system), these 'coordinated' computing systems offer reduced incremental cost, modularity, better reliability, response and performance, and improved utilization. Thus, the feasibility and realization of coordinating computing opened avenues to challenging applications and research problems.

There has been a growing interest in developing strategies and methodologies for various applications and problems of coordinated computing. Distributed control, task decomposition, synchronization, query processing, performance analysis, and distributed operating system design are prominent of these. Design and analysis of algorithms for a coordinated computing system, generally called 'Distributed System', form the backbone for efficient utilization of such a system. A distributed system is a

collection of heterogeneous computers interconnected by a communication network. The theory of algorithms traditionally deals with the design and analysis of algorithms on abstract machine models, such as the Random Access Machine (RAM) or the Turing Machine [1]. Algorithms on these stored program machines are characterized by computational parameters such as time complexity or space complexity. However, such characterizations do not hold in the domain of distributed systems. Studies on algorithms for distributed systems have been made by several researchers in recent years; the books edited by Van Leeuwen and Raynal [38, 45] give a comprehensive treatment on distributed algorithms for various applications.

The main focus of this dissertation is to design efficient algorithms in the domain of distributed systems, for two generic problems, namely the distributed network protocols and distributed graph problems. In particular, we concentrate on the design of distributed network protocols for graph traversals, construction of spanning trees, and for obtaining topological configurations of a distributed system. Under the distributed graph problem domain, we design new algorithms for finding important topological parameters such as centers and medians. We also show a counterexample for the only other algorithm for finding centers and medians in the literature, and provide corrective solutions to it. We design an algorithm for the graph recognition problem, particularly to the mesh (grid) recognition problem, which is the only algorithm to be found so far in the literature. Finally, we show an application of some of these algorithms for certain problems in distributed sensor networks. We believe that these new algorithms pave the way to the design of more efficient algorithms for several other related problems well known in the literature.

1.2. Distributed System

A distributed system, informally, refers to the integration of autonomous computing systems that cooperate to achieve a common goal. Although enormous research and development have taken place over the last decade in this area, there is no universal definition describing a what a distributed system is. However, all "formal" definitions found in the literature mean the intuitive definition above. The following statements that define a distributed system encompass most of the definitions:

> *A distributed system is one in which the entities that form the system cooperate in achieving a common goal by exchanging messages. The autonomous processing elements with/without local data are loosely coupled with a shared communication system, without any global control or shared memory.*

Another view expressed by Enslow [18] considers the decentralization of control, processors, and data. It also requires that distribution is transparent and system users need not have the knowledge of the various processors of whcih the system is made. In figure 1.2.1, we illustrate the various types of distributed systems described by Enslow. Each axis in Enslow's cube model contains points in increasing order of decentralization. As seen in the model, a collection of autonomous heterogeneous processing units with no global data that cooperate in achieving a common task is classified as a *fully distributed system.* Enslow's model provides a basis for studying various aspects of distributed systems.

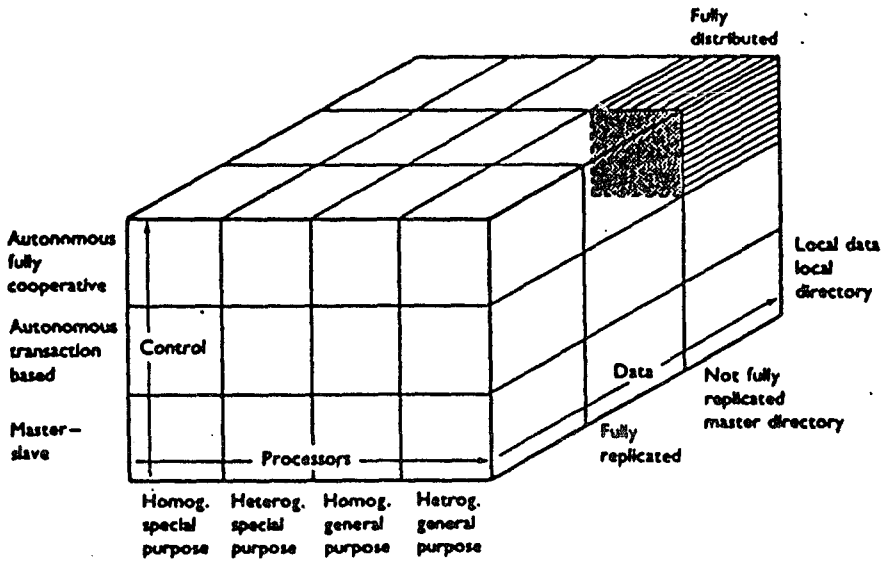

Figure 2.1 Enslow's model of distributed system types

Lynch and Fischer [41] describe a mathematical model for distributed systems and a model for their input and output behaviors. Both the models are set theoretic, built from the standard mathematical constructs such as sets, sequences, functions, and relations, rather than axiomatic models consisting of lists of desired properties of systems.

One of the fundamental characteristics of distributed systems is that the inter-entity message transit is subject to variable delays and failures, unlike in the tightly coupled systems. This arises from the fact that the system is loosely coupled via a communication network and delays due to congestion and traffic occur besides the propagation delay. These factors govern the efficiency of any algorithm on a distributed system.

1.3. Distributed Algorithm

Consider a distributed system where a set of physically distinct computational units work on a common problem while their operation is coordinated via communication channels connecting some or all of these units. Each computing unit has certain processing and memory capabilities, and is preprogrammed to perform its part of the computation, as well as send and receive control/data messages over the communication links. The program residing on each node is called the *distributed algorithm* [48].

Generally, distributed algorithms are complex since there is no global control over the system to control the algorithm behaviors, unlike in sequential or parallel computing. It is assumed that the same copy of the algorithm runs on all the constituent processors of distributed systems and the goal is achieved by sending/receiving messages and by the use of proper protocols to extract/embed information from/to the messages. Regardless of the manner and degree of distribution, the following two fundamental assumptions are common to all distributed algorithms.

1. Each node has only a partial picture of the total system and its decisions are bound on this information.

2. There exists no system-wide common clock.

A fully distributed algorithm is characterized by the following properties.

1. All nodes have an equal amount of information.

2. All nodes make a decision based solely on its preprogrammed local information.

3. All nodes bear equal responsibility for a final decision.

4. All nodes expend equal efforts in effecting a final decision.

5. Failure of a node, in general, does not result in a total system collapse.

Distributed algorithms for a number of problems have been reported in literature. Two classes of problems considered fundamental namely network traversal and global state determination, are discussed by Raynal [45]. Distributed for a variety of problems such as distributed mutual exclusion, leader election, termination, failure detection, synchronization, and graph algorithms (shortest path, minimum spanning trees, finding centers and medians, connected components, graph recognition etc.) have been extensively studied and reported in the literature [50, 20, 3, 49, 34, 12].

1.4. Contributions of the dissertation

We consider two generic problems in distributed systems, namely the network traversal and related applications; and distributed graph algorithms. Network traversal is a vital activity in networks, involving a wide variety of applications. We present distributed algorithms for traversing a network and constructing spanning trees. Depth- first traversal is fundamental to the development of several other algorithms besides being extensively used to initialize the systems, give system identities etc. We present an efficient algorithm to construct depth-first-tree, distributively. The algorithm is shown to be efficient for both weighted and unweighted networks.

In distributed systems, no global information is maintained at each node since the system is dynamic and the global information might require prohibitive space to

store the information. However, each node is equipped with algorithms that will enable these nodes to determine certain topological parameters, whenever necessary. These algorithms are called *learning algorithms*. We present a set of learning algorithms for finding connectivity, biconnected components, and global topology. These are shown to be the applications of distributed depth-first-search algorithms.

Locating centers, medians and centroids play an important role in the design of efficient algorithms for certain applications. We show that the median and center finding algorithms of Korach, et al. [34] fail to terminate and arrive at a correct solution, for certain tree configurations. We present two basic algorithms for locating centers and medians and show that these algorithms are message optimal. These basic algorithms could also be used to find centroids, diameter and similar properties of networks.

Its well known that *a priori* knowledge of network topology leads to designing efficient solutions to certain problems in distributed systems. Graph recognition in the domain of distributed systems was first proposed by Ramarao [42] who presented algorithms to recognize simple structures such as rings, trees, complete graphs, bipartite graphs, and stars. We solve an open problem, i.e., recognition of grid (mesh) structure distributively and present efficient algorithms.

Distributed sensing is an important application area of distributed problem solving. We study application of some of the algorithms discussed above for analyzing communication and reliability issues in distributed sensor networks.

1.6. Scope and Organization of the Dissertation

Basic concepts of distributed computing are developed in chapter 2. We explain measures of computational complexities for distributed algorithms. In this chapter, we enlist the fundamental problems in distributed computing and summarize some interesting results.

We present our distributed algorithms in two parts. The first part (chapters 3 and 4) deals with the distributed network protocols (DNP), while the second part focuses on distributed graph algorithms. In chapter 3, we present a set of algorithms addressing the generic problem - graph traversals. We present a family of algorithms for broadcast, spanning tree construction, depth-first and breadth-first traversals of a network. We emphasize upon our results on depth-first-search, and spanning tree constructions. Applications of our traversal algorithms for various network learning and computation problems are discussed in chapter 4. These include algorithms for finding connectivity, biconnected components, network topology.

We present efficient algorithms for finding centers and medians in chapter 5, besides providing corrections to the only other algorithm in the literature, by Korach, et.al. [34]. Extensions of these algorithms to weighted networks are also discussed in this chapter.

Motivation for graph recognition is developed in chapter 6. We present our algorithm for recognizing grid structures in this chapter.

Issues in distributed sensor networks are presented in chapter 7. We present algorithms to study the communications and reliability problems in distributed sensor networks. We conclude with a discussion of open problems in chapter 8.

CHAPTER 2

PRELIMINARIES

2.1. Parallelism and Distribution

Information processing that focuses on the parallel manipulation of data belonging to one or more processes/processors, to solve a single problem, is generally acknowledged as *parallel processing*. Often, a single task is decomposed into independent subtasks and each subtask is executed on individual processing elements simultaneously. Parallel algorithms refer to the algorithms that are executed on a parallel computer, which is generally a tightly-coupled multiprocessing system with shared memory or a pipeline architecture. In any parallel execution, synchronization and communication are the two important mechanisms to resolve contentions for resources in order to subsequently achieve a global solution. It should be noted that synchronization and communication are mutually dependent; synchronization can be achieved if communication mechanism exists and conversely.

Distributed systems are characterized by the absence of shared memory. Hence, parallel execution in the context of distributed systems defines a new set of issues involving synchronization and communication mechanisms that cannot be handled by synchronization and memory access primitives. Algorithms on distributed systems must involve the concept of *message* and hence the primitives must be defined for sending and receiving messages. All distributed systems use these mechanisms to achieve synchronization and control. It is important to note that the behavior of a dis-

tributed system and communication by messages can be simulated on a centralized system, at a reasonable cost.

We shall now see the basic differences between the distributed and the non-distributed (sequential and parallel) algorithms. The control information is available at one node in a sequential or parallel system, to exercise control over the algorithm execution. In a distributed system, different nodes cooperate by exchanging local control information, and no single node has complete control information. Due to unpredictable delays in communication the order of computation and hence operational results become unpredictable during the execution of distributed algorithms; unlike in the nondistributed systems. Event control and execution take place at one site in parallel or sequential systems (mainly due to their architecture) unlike in distributed systems. Finally, the performance of sequential algorithms is characterized by processing time and the number of processors in parallel computing. Processing time is considered negligible in the analysis of distributed algorithms since the communication costs are far higher, and hence the number of messages required to execute an algorithm is a measure of the efficiency of a distributed algorithm. In table 2.1 [13] we summarize the basic differences between the distributed and nondistributed algorithms.

Table 2.1

Algorithm	Nondistributed	Distributed
1. Control information availability	central	different sites no complete information
2. Order of operational results	predictable	unpredictable
3. event control and execution	at one site	different sites, no shared memory
4. performance measures	Time complexity space complexity	time complexity message complexity

2.2. Features of Distributed Algorithms

Distributed algorithms are characterized by the fact that control is distributed throughout an algorithm. There is no fixed hierarchical or prioritized relation among the set of processors over which the control is distributed, i.e., there is no master process that controls the algorithm or the system. It is this situation where each process is as powerful as the other in the system that makes the design of distributed algorithms extremely difficult. We now consider some features of distributed algorithms that must be considered while designing new algorithms [44].

a. Degree of partitioning

An important feature of a distributed algorithm is the extent to which an algorithm is partitioned, a concept related to the symmetry of the roles played by different processes. There are several levels of symmetry:

i. Asymmetry: every processor executes a different text, so no two processors can be interchanged.

ii. Textual symmetry: the text executed by each process is the same, but the names of each process is distinct. So, the symmetry could be broken by the use of these names.

iii. Strong symmetry: the program texts are identical, and there is no reference to process names. Identical or different behaviors are possible, according to the received messages.

iv. Total symmetry: the program texts are identical and all the processes behave in the same way.

It is clear from the definitions above that asymmetry denotes the lowest level and total symmetry represents the highest level of degree of partitioning of control. However, most of the distributed algorithms need the names or identities (ID) of processes to be distinct to resolve conflicts. For example, if a process x receives messages from two nodes i and j, it might resolve any contention by selecting the message from the node with largest ID.

b. Fault-tolerance

As discussed earlier distributed system does not collapse under some node/link failures. Hence, it is imperative that the distributed algorithms are made resilient to failures. Since no node has any special control, these algorithms must be designed to continue to function in spite of failure.

c. Network Models

It is desirable to design algorithms with least possible assumptions regarding the system. However, this depends on the application and the generality of the system. Fewer assumptions on the system leads to sophisticated algorithms.

d. Algorithm Complexity

Another desirable feature of a distributed algorithm is the use of fewer messages to achieve the desired goal at the end of the algorithm. Efficiency of an algorithms is measured by the total number of messages exchanged during the execution of the algorithm. Small number of messages lead to less traffic in the network and hence less delay.

e. Global state

The ability of any process to take decisions without needing a knowledge of the global state is another measure to evaluate an algorithm. It is well known that the absence of a system-wide clock inhibits any node having the global knowledge.

2.3. Paradigms in Distributed Computing

In a sequential computing environment, we are familiar with the well known paradigms for problem solving, such as *divide and conquer or dynamic programming.* In this section, we briefly discuss some paradigms associated with programming distributed systems [47].

1. **Agreement and Commitment:**

This paradigm deals with establishing agreement in a distributed system. The problem is to design a protocol by which all processors agree upon a value. This value may be a data, or the outcome of some computation, or a value that is used to effect a decision.

Agreement problem is simple when the processors do not fail. However, when a processor is faulty, there is ambiguity about the *integrity* of the data that is sent from it. In such an environment, the agreement problem is to devise protocols in which a value known to a designated processor called the *transmitter* is disseminated to other processors such that:

a. all non-faulty processors agree on some value.

b. if the transmitter is non-faulty, then all non-faulty processors use its value as the one they agree upon.

There are a number of variants of the problem that depend on whether processor failures are detectable whether processor speeds are bounded and whether messages can be authenticated. Several protocols have been devised and reported in the literature. One of these is the broadcast protocols which will be discussed in chapter 3. Broadcast protocols describe how a message being broadcast is to be disseminated to the processors in the network. Byzantine agreement deals with the agreement problem when no assertion can be made about the behavior of faulty processors.

In a distributed system, it becomes necessary to coordinate a collection of actions at different processors so that all actions occur or no actions occur. For example,

when a file is replicated at different processors it is necessary to see that all copies of the file remain identical. If a change is effected on a file it must be done on all or none. A *commit protocol* is an agreement protocol in which the value agreed upon is computed by applying a *decision function* to the values of the participants. Thus, a commit protocol establishes:

i. all non-faulty processors agree upon some value.

ii. the value agreed upon is the result of applying a decision function to the values of all participants.

It is clear that given an agreement protocol and a decision function, it is simple to construct a commit protocol.

2. The state machine approach:

A sequential process could be easily expressed in terms of a finite state machine which reads its input, performs some known computations, generates some output, and transits to some state. Any concurrent program can be represented as a set of concurrent processors that interact by sending requests, and awaiting replies from a state machine. processors are the source of input to the state machine as well as the destination for its output.

It is clear that in a distributed system, when a node receives a message, it can determine the past state of the sender which is not necessarily the current state of the sender. Consequently, the state of the system is not known to any process. When a concurrent program is viewed in terms of a state machine, a process makes a request of the state machine whenever

i. it needs to be delayed until the system state is conducive to continued execution,

ii. it has changed the system state, and consequently there might be delayed processes that can continue executing, or

iii. it needs information about the state of the system to determine how to proceed.

Using i and ii above, the state machine can synchronize the processes and use of iii allows the state machine to control what processes do. The state machine approach for systems in which arbitrary behavior could result from a failure has been discussed in [47]. Many synchronization protocols for distributed systems can be derived by using the state machine approach and then applying application dependent optimization.

3. Computing global states:

This paradigm is employed to solve a number of problems in distributed computing. By computing the global states it is possible to detect the invariant properties in the system, depending upon the application. For example, "the total number of tokens in the system is n" is an invariant predicate, or "the system is deadlocked" is an invariant predicate. So, by devising an algorithm to compute the global state of a distributed system, one could solve a number of problems. Since the system states keep changing, its possible to take a *snapshot* of the system and compute the invariants [11].

4. Elections:

Election paradigm is used to make decisions in a computing system where a set of processes must choose one outcome from among some candidates, any of which is

acceptable. The decisions could be to choose a coordinator (break symmetry!) in the system, or to resolve a conflict (decide which of the two or more contending processes should succeed in the action being attempted).

Elections are implemented in several ways in distributed systems. There are a variety of variations of elections considering several topological configurations for the system and the link/node reliabilities, weighted voting, etc. An extensive treatment on election algorithms can be found in [21].

2.4. Concepts and Techniques

The distributed algorithms make use of standard techniques such as acknowledging messages, broadcasting to some processors, etc., irrespective of the paradigm being employed to solve a problem. We consider three important techniques often used in distributed algorithms.

1. **Diffusing Computations:**

   This principle is used in algorithms where a particular type of control is to be exercised, and was proposed by Dijkstra and Scholten [17]. Mainly termination detection of a process or mutual blocking of two or more processes use this principle to the spanning tree of the graph representing the network. The *root* of the tree plays a different role from the other nodes in the network; it sends information to all other nodes and receives replies from other nodes in the following sequence:

   i. the root sends a message to its neighbors in the spanning tree.

ii. on receiving the message, each node propagates the message to its children and waits for a reply. Upon getting the reply, it sends reply to its parent. If it is a leaf, then it sends the reply to its parent.

iii. algorithm terminates when the root receives reply from all its neighbors.

2. Circulating token:

The *token* is an special message which is generally circulated in a network. A processor in the network will need to possess the token before it can become 'privileged' to execute some special action such as accessing a shared resource in the case of the mutual exclusion problem. This is a simple but powerful technique in solving important problems such as mutual exclusion and termination detection. This principle is extensively used in the design of ethernet.

3. Time stamping:

This principle makes it possible to order the events in a consistent manner in relation to the interactions between the processes. It is used extensively to resolve contention and form some kind of ordering at a process level. It is already said that total ordering of events in a distributed is not possible. Lamport [37] introduced the concept of 'logical clocks' which is the local clock of each processor P_i, to define the local time; by an integer, say h_1. h_1 is initialized to 0 and its successive values form an increasing sequence. Every message m issued by P_i is stamped with the current value of h_i and process ID, to form the tuple (m, h_i, i). Each process P_i manages its clock as follows:

i. when P_i receives a message (m, h_j, j), it sets h_i it sets h_i to max(h_i, h_j) + 1.

ii. when P_i issues a message (m, h_i, i), the value of h_i is incremented.

iii. h_i can also be incremented between pairs of events internal to P_i.

This mechanism provides a partial ordering of events. It is possible to derive a total ordering from this by means of topological sort.

CHAPTER 3

NETWORK TRAVERSAL

3.1. Introduction

Many applications involving distributed systems require traversal of messages through the underlying network. As explained in section 2.5 a distributed system can be viewed as a graph with the vertices representing the processors and the edges representing the communication links. Traversing a graph has been studied exhaustively with respect to sequential systems. Since message passing is the only mechanism in implementing such traversals in a distributed system environment, it is essential to study techniques related to network traversal. Independent of the nature of computations, the traversal protocols are of great importance and some of these form the basic building blocks for the design of some complex algorithms. It will be seen later that construction of a spanning tree for a graph underlying a distributed system is a basic design strategy to solve several problems efficiently. In this section, we present algorithms for network traversal. Information dissemination by broadcast is also an essential activity. We study algorithms for broadcasting information in the network and constructing spanning trees of a graph. We present algorithms for constructing trees in depth-first and breadth-first fashion that parallel their counterparts in sequential and parallel computations. We briefly discuss the construction of minimum spanning trees.

All the algorithms discussed in this section are fundamental in nature and find extensive applications in distributed systems. For all these algorithms, we assume the standard model for the distributed system unless specified otherwise. We discuss the algorithms for constructing spanning trees and those for building depth-first-search tree in detail. However, discussion on broadcast, minimum spanning tree, and breadth-first- search tree is brief and is limited to stating some important results.

3.2. Broadcast Algorithms

Broadcast is one of the most fundamental tasks in distributed computing.

Definition: *Broadcast* is the delivery of a message from a processor to *all* other nodes in the network. The processor initiating broadcast is called the *source* and the other processors are called *destination* nodes.

In contrast to broadcast, the term *multicast* refers to the delivery of messages to some specified subset of all the processors in the network. The problem of broadcast has been investigated by several researchers for a wide range of applications [57, 16, 5, 59, 28]. These applications include broadcast routing, topology broadcast, sorting etc. The communication and information tradeoffs have been studied by Awerbuch et.al [5] and bounds for broadcast are established when each node knows the topology of the network in the radius row ≥ 1 from itself. Strategies and protocols for performing fault-tolerant broadcast can be found in [5, 40].

3.2.1. Strategies for broadcast:

a. Flooding:

In this scheme, a message is sent to all other nodes by "flooding" the links of the network, i.e., by propagating the message over all the links.

b. Tree broadcast:

This scheme assumes that there exists a spanning tree of the network and that each node is aware of its parent and children in the spanning tree. Then, the message from a node is "broadcast" to the other nodes by forwarding the message over the tree edges only.

It is clear that flooding requires $\Theta(e)$ messages whereas the tree broadcast needs n-1 messages where e and n denote the number of links and nodes in the network, respectively. However, for tree broadcast we need a spanning tree constructed a priori. It is interesting to note that the problem of constructing a spanning tree from a node requires Θ (e) messages.

Complexity bounds:

We now present the complexity bounds for the broadcast problem. The proofs are left out and can be found in [5].

Lemma 3.2.1.:

The broadcast problem by flooding has a message complexity of $\Theta(e)$ whereas that by tree broadcast is $\Theta(n)$. □

Lemma 3.2.2.:

Broadcast by flooding requires at most O(d(G)) units of time whereas tree broadcast requires at most $O(h_{max})$ time units where d(G) is the diameter of the graph G

and h_{max} is the maximum height of subtrees in the spanning tree rooted at the source. □

Also, it is shown in [59] that all-to-all-node broadcast by flooding has a worst case time complexity of $\Theta(n^2)$.

3.3. Construction of spanning trees:

The problem of building a spanning tree for a network is a fundamental problem in the area of distributed algorithms. Specifically, it is of great advantage to be able to maintain a network as a tree. We saw in the previous section that communication cost for broadcasting message can be reduced from O(e) to O(n) by employing tree broadcast. A large number of important problems in distributed computing could be reduced to the problem of finding spanning trees, details of which are described in the following sections. In this section, we present algorithms for constructing an arbitrary spanning tree of a network.

We present a simple centralized algorithm for constructing a spanning tree which is similar to the broadcast algorithm. The algorithm is explained informally as follows. A similar algorithm is described in [48].

3.3.1. Algorithm SP_T:

Input: Graph G(V,E) underlying the network as in a standard model.
Output: Each node knows its neighbor in a spanning tree T of G.
Messages: FIND and RETURN

i. The source node broadcasts the FIND message to all its neighbors.

ii. Every node i, upon receiving the FIND message for the first time from its neighbor j, marks j as its parent in the tree T; and sends the FIND message to all its neighbors except j. It ignores all subsequent FIND messages, if received.

iii. If the FIND message is received from all its neighbors, then it sends a RETURN message to its parent node.

iv. Each node upon receiving the RETURN message, marks the node as its child node; upon receiving messages from all its neighbors, it sends RETURN message to its parent node.

v. Algorithm SP_T terminates when the root node receives messages from all its neighbors.

Theorem 3.3.1: The algorithm SP_T constructs a spanning tree of the graph G and terminates in finite time, after which each node knows its neighbors in the tree T. □

Theorem 3.3.2: The algorithm SP_T builds a spanning tree T of the graph G using exactly 2e messages in at most 2d(G) units of time, where d(G) is the diameter of the graph G. □

Designing decentralized algorithms for constructing spanning trees is a complex problem. These algorithms could be derived from the minimum spanning tree algorithms of Gallager et.al. [20].

3.4 Distributed Depth-first-search

The problem of depth-first-search (DFS) on graphs has been extensively studied in the context of sequential algorithms [1, 26]. DFS being a simple and efficient traversal procedure, finds enormous applications such as finding biconnected components, calculation of maximum flow, detecting planarity of graphs, etc. In this section, we study a distributed version of DFS and present distributed algorithms.

Many applications in distributed systems require traversal of messages through the underlying communication network. The problem of distributed depth-first-search is defined as follows.

Problem: Consider a communication network. The aim is to equip the set of processors in the network with a control algorithm which will allow a processor P_i in the network to effect a depth-first-traversal through the graph underlying the network using messages. The output of the algorithm is a depth-first-search (DFS) tree of the communication network kept in a distributed fashion, i.e., at the end of the algorithm each node will know its neighbors in the DFS tree.

Recent papers [3, 36, 14] present improved algorithms for distributed depth-first-search (DDFS) for graphs representing communication networks. Earlier work of Cheung [13] presents a DDFS algorithm with message and time complexities of 2e for both where e is the cardinality of the set of undirected links in the graph.

Awerbuch [3] improves the time complexity of Cheung's algorithm and presents an algorithm with time complexity less than 4n, requiring 4e messages where n is the cardinality of the set of nodes in the graph. Lakshmanan, et.al., in [36] show lower bounds for message and time complexities and present an algorithm that is time optimal requiring less than 4e - (n -1) messages. In [14], both the time and communication complexities are improved with the communication cost shown to be less than 3e messages and the time complexity being 2n.

Before we describe our distributed depth-first-search algorithm [50, 35, 51] we discuss certain properties of DFS that reflect on the performance upper bounds of any algorithm for DFS. Reif [46] presents interesting results concerning the complexity bounds for DFS ordering. The paper by Reif demonstrates that DFS is inherently sequential and hence, it is concluded that DFS cannot be speeded up beyond linear complexity using a polynomial number of processors. It is shown that DFS ordering is P-Complete.

Theorem: (Reif)

DFS-order is P-Complete. □

While designing our algorithm, we take this fact into consideration. We observe that despite traversing the edges of G in parallel, the algorithm cannot be speeded up beyond O(n) time complexity.

We now present a new algorithm for DDFS with time and communication complexities of O(n) for both. The algorithm is shown to use exactly 2n-2 messages and to terminate after 2n-2 units of time. This reduction in message complexity is

achieved by the effective use of extended message format in the algorithm and it is later shown that the total number of bits used for communication is less than that used in the earlier algorithms for certain networks.

3.4.1 The model

The communication network is represented by the graph G(V,E) where V and E are respectively the sets of vertices and undirected edges. We assume a standard model for the network as explained in chapter 2, except that we assume each node knows its neighbor IDs.

3.4.2 Algorithm DDFS:

In this section, we describe the extended message format and present the algorithm. A formal description of the algorithm is given in appendix A3.4.

3.4.2.1 Message Format:

There are two messages namely START and DISCOVER. The DISCOVER message has the header and an appendage of n bits at most. The appendage in the message is a bit array such that the i^{th} bit represents the state of the i^{th} processor. The state of a processor with respect to the appendage is either VISITED or UNVISITED. The START message contains the start header and is used as signal to initiate the algorithm.

3.4.2.2 Informal description of the algorithm

In the earlier algorithms the basic idea was to select a node as the son at every 'center of activity' and send the 'VISITED' message to the rest of the unvisited

neighboring nodes, in parallel. Each node waits for an ACK message to be received from all the neighbors, after sending the VISITED message to them.

Our algorithm eliminates the need for a separate VISITED message and this information is embedded in the message itself. The algorithm operates as follows. A node is initially given the START message from the external world. The START message signals the start of the algorithm and the node that receives the START message becomes the root node of DFS.

The root node forms the DISCOVER message with a bit-array of size equal to the maximum of the root i.d.(name) and the largest integer denoting its neighbor i.e., if k is the largest ID of the nodes adjacent to the root and r is the root ID then max(k, r) bit-array is created. Every bit in the bit-array is initialized to UNVISITED state. The root node selects one of its neighboring nodes as its son, marks its (root) position in the bit-array of the message as VISITED, and sends this message over the link to the selected son. It may be recalled that since each node i is identified as integer i, $1 \leq i \leq n$, position of each node in the bit-array is unique. Clearly, the contents of the bit-array depict the prevalent state (VISITED/UNVISITED) of nodes in the network at any instant of time.

Upon receiving the DISCOVER message for the first time, each node i does the following.

i. It marks its position (i^{th} bit) in the bit-array of the received message as VISITED.

ii. It marks the node from where the DISCOVER message was received, as its father in the DFS.

iii. It extends the bit-array to k bits if k > size of the received bit-array where k is the largest ID of its neighbors and initializes the added bits to UNVISITED state.

iv. The node chooses an UNVISITED node as its son from its adjacency list, if one exists and transmits the DISCOVER message with the updated bit-array to the selected son node. To choose an UNVISITED neighbor, next node in the adjacency list of node i is taken and the state of the chosen node is checked in the received bit-array; if found VISITED, the process is repeated with other nodes in its adjacency list. If all the neighboring nodes are already VISITED, i.e., node i is a leaf node in DFS, the received DISCOVER message with updated bit-array is returned to the father node.

If the DISCOVER message is received from a son node, then only step iv above is executed. The algorithm terminates when the root node receives the DISCOVER message with all nodes in its adjacency list exhausted in the search.

If the total number of nodes in the network is known *a priori*, then the root node creates a bit-array of size n-bits and no other node in the network need extend the bit-array.

3.4.3.Complexity Analysis and Proof of Correctness.

In the previous section, we presented the DDFS algorithm employing two messages with the above-described message format. In this section we prove that time and message complexities of the algorithm are both O(n). We also show that the algorithm is optimal in message complexity and further demonstrate that DDFS has a message complexity lower bound of O(n). We also establish that the communication complex-

ity of our algorithm is less than that reported in [14] in terms of total number of bits used for communication despite increasing the message size in our algorithm.

Theorem 3.4.3.1.

DDFS has the message complexity lower bound of O(n) for n > 1.

Proof

This bound is easily seen. It is clear that at least ONE message must be sent from any node i to any node j in the network, i,j ∈ V, since no memory is shared in the network. Hence at least n-1 messages must be sent over the links so as to communicate to all the n nodes at least once. Hence DDFS has an O(n) lower bound in message complexity. □

Theorem 3.4.3.2: The algorithm is optimal in communication complexity and uses exactly 2n - 2 messages.

Proof

Every node in the network except the root node receives only one DISCOVER message from its father (forward path) and sends one DISCOVER message to its father (return path) . Also, it is clear from the algorithm that DISCOVER message is not sent to an already VISITED node in the network. Thus, each of the n - 1 nodes (excluding the root) exchanges DISCOVER messages with its father exactly twice. Hence, the total number of messages used in the algorithm is exactly 2n - 2. Clearly, the message complexity of the algorithm is O(n) and is optimal within a constant. □

Theorem 3.4.3.3: The algorithm terminates after 2n-2 units of time if every message

is delivered in one unit of time and is time optimal.

Proof

The total time needed for the algorithm to construct DDFS is the time required to transmit the messages over the links. The total time needed to transmit a total of 2n - 2 messages is 2n -2 units of time if all messages are delivered in at most one unit of time.

It is shown in [36] that distributed algorithm for DFS has a worst case time complexity of 2n - 2 and hence the algorithm is optimal in time. □

We now compare our algorithm with the algorithm in [14] for bit-wise message complexity. For convenience we call our algorithm as A1 and that in [14] as A2. Let n be the total number of nodes in the network and m be the number of bits transmitted for each message of A2.

Lemma 3.4.3.4: The total number of bits used for communication in the algorithm A1 is less than that for the algorithm A2, for $n > 1$ and $m > 4$ bits.

Proof

In algorithm A1, the number of bits transmitted for the DISCOVER message is at most (m+n) bits due to the possible n-bit appendage to the message of A2. Total number of messages that are used in A2 is $< 3e$ where $e \leq n(n-1)/2$. Hence, for a fully connected network, total number of bits used for communication is $< 3n(n-1) * m/2$. In A1, the total number of messages used is 2(n-1) and hence the total number of bits used for communication is at at most 2(n-1) (m+n). We derive the condition for m and

n such that

$$2(n - 1)(n + m) < 3n(n - 1)m / 2.$$

$$\text{i.e.} \quad 4(n + m) < 3nm$$

$$\text{i.e.} \quad 4m/(3m - 4) < n.$$

For $m > 4$, we get $n > 1$ for the inequality to be satisfied and hence the proof.□

In table 3.1 we summarize the performance of algorithms for DDFS. However, it should be noted that the message size of our algorithm is greater than the others in the table.

Table 3.1

Author	Year	Time Complexity	Communication Complexity
T.Cheung	1983	2e	2e
B.Awerbuch	1985	< 4n	4e
K.B.Lakshmanan, N.Meenakshi and K.Thulasiraman	1987	2n - 2	< 4e -(n - 1)
I.Cidon	1988	≤ 2n	< 3e
M.B.Sharma, S.S.Iyengar and N.K.Mandyam	1988	2n - 2	2n - 2

3.4.5 Summary:

We have presented a new DDFS algorithm that is optimal in message complexity. The algorithm is shown to use exactly 2n - 2 messages with the time complexity of 2n-2 units of time. We have shown that the extended message format reduces overall message complexity in terms of total number of bits used for communication. It may be noted from table 3.1 that although the message complexity of our algorithm is O(n) the number of bits per message in our algorithm is more than that in the other algorithms in the table. It is interesting to see that the proposed algorithm performs the same way for both synchronous and asynchronous networks since the 'center of activity' moves sequentially in a deterministic manner.

Consider a weighted network where the edges of the graph G representing a network are labeled with non-negative weights. Weight of an edge represents the cost of communication along that edge. In the depth-first traversal schemes of all the previous algorithms each edge is traversed at least 2 times, irrespective of the cost of communication. The advantage of our algorithm is that DDFS can be performed by traversing over edges with least communication cost, thus avoiding expensive traversals over all the edges.

3.5 Distributed Breadth-first-search Algorithms:

As in the case of DFS, breadth-first-search (BFS) is also defined in the context of distributed systems. The problem is similar to the minimum hop problem defined in communication networks. In this section, we present results on distributed BFS. The problem is stated as below:

Problem: Given a graph G(V,E) and given a particular node r, find a breadth-first-search tree rooted at r. At the end of the algorithm each node knows its neighbors in the BFS tree.

Distributed breadth-first-search determines shortest paths from the root to every other node in the connected component of r. The Bellman-Ford algorithm [25] computes the min hop distances and is used for routing in Arpanet. This simple algorithm is expensive in terms of communication complexity of O(n e). The algorithm of Frederickson [19] performs DBFS using O(n. $\sqrt{e}$) messages. The algorithm of Zhu and Cheung [65] determines BFS distributively and has time and message complexities of O(n^2), both. The algorithm of Awerbuch et.al. [4] gives a near optimal distributed algorithm for BFS. The algorithm uses a novel concept of defining multiple synchronizing nodes. Synch nodes are defined for every $n^{.2}$ levels. This algorithm is shown to have message complexity of O($n^{1.6}$ + e) and is optimal for networks with e $\geq n^{1.6}$.

3.6 Minimum spanning trees

The problem of constructing minimum spanning trees for weighted asynchronous networks has been widely studied in literature [6, 20, 15]. The technique of implementing Kruskal's algorithm for finding minimum spanning trees (MST) has been used by Gallager et. al. [20]. This algorithm is of great importance in the area of distributed algorithms since the technique of "divide-and-conquer" has been defined in the domain of distributed systems. In fact, the *distributed divide and conquer* paradigm is used in the design of algorithms for several other important problems in

literature.

The basic principle of the algorithm is explained as follows. Define a *supernode* as a subnetwork on which the problem under consideration is already solved. Now the idea is to choose one and only one appropriate neighbor of a supernode and merge these two into one large supernode. Eventually, when the algorithm terminates there will be exactly one supernode that contains the solution for the entire network. In order to reduce the merge costs a novel technique has been adopted. Each supernode maintains a level information with it and the level is a measure of its size. A supernode at level k contains at least 2^k nodes in it. When a supernode S' merges with a neighbor supernode S and level(S) > level(S') , then S' simply merges with S, but the level of S is unaffected. But, level(S') is changed to the level of S. If level(S) < level(S') then S' waits until level(S) reaches its level. If the two levels are equal, then the two supernodes merge and the level of the merged supernode is incremented by one. In order to reduce communication costs during the merge process, each supernode maintains its spanning tree.

The communication complexity of this algorithm is shown to be at most 5n log n + 2e, whereas the time complexity is O(n log n). An optimal algorithm for MST given by Awerbuch [6] has time complexity of O(n) and message complexity of O(e+nlogn) and is the best known result in literature. The algorithm employs a slight variation of the merge process described above.

CHAPTER 4

NETWORK LEARNING ALGORITHMS

4.1 Introduction

In a distributed system, global information is not maintained at each node due to the volatile nature of the network as described in section 1.4 of chapter 1. Each node then, is equipped with algorithms such that they can determine topological parameters and such class of algorithms are called *Learning algorithms* [48, 2, 45, 34, 49]. In this chapter, we present algorithms for determining certain network parameters. In particular, we discuss algorithms for finding connectivity, global topology, and biconnected components. These algorithms are obtained mainly as application of distributed depth first search algorithms described in 3.4. In fact, any traversal algorithm could be used to determine these parameters. However, we show the advantages of employing the DDFS algorithm described in [50].

4.2 connectivity algorithm MCT:

The problem of finding connectivity was studied by Segall [48] and a set of algorithms were presented for this problem.

Problem: Design a distributed algorithm such that upon execution of the algorithm, each node will learn the nodes that are connected in to it in the network and each node will know of algorithm termination. The latter part of the problem deals with the termination property of distributed algorithms.

In other words, the problem is to determine the set C of nodes such that for any two distinct nodes i,j∈ C, there exists a path between them. A range of algorithms (CT1 through CT5) were presented by Segall [48] to determine connectivity. The main idea behind these algorithms is to use a spanning tree construction algorithm and collect the node IDs in the process of tree construction. Communication complexity of the simplest version of the protocols CT1 is shown to be 2nelog N where N is the maximum integer ∈ Z from where the names are chosen. Algorithms CT2 through CT4 that incorporate termination property have very high communication costs of n(2e+n+1)logN bits. Last in the range of these algorithms CT5 a variation of the earlier algorithms is shown to have communication complexity bounded by 2nelogN bits.

We now describe our algorithm MCT which has a communication cost of 3n(n-1) bits. This algorithm is based on the DDFS algorithm described in section 3.4.

Algorithm MCT:

Algorithm MCT runs in two phases.

1. Construct a depth-first-search tree distributively using the DDFS algorithm in section 3.4. The bit vector at the termination of the DDFS algorithm depicts the nodes that are connected in the network. All those bits marked VISITED represent the connected set of nodes. Names of these nodes is extracted from the bit vector.

2. In this phase, the root node of the DFS tree constructed in phase 1 transmits the bit vector to all other nodes in the tree. A message called CONNECT

with a message structure similar to that of DISCOVER message of DDFS algorithm is transmitted from the root to all other nodes in the network using the tree edges. A node, upon receiving the CONNECT message learns the nodes that are reachable from it from the bit vector and forwards the CONNECT message to its children in DFS tree.

Formal description of the algorithm is in appendix A4.2.

4.2.2 Complexity Analysis:

Theorem 4.2.2.1:

Suppose node s receives the START. Then

(a). Algorithm MCT correctly determines all the nodes connected to the network containing node s.

(b). All nodes learn the names of the connected nodes in the network containing node s in finite time.

Proof:

The proof for (a) follows from the fact that the connected components of a graph could be determined using DFS and it has been shown that the DDFS algorithm finds DFS tree correctly. Part (b) is proved as follows. By theorem 3.4.2, DDFS terminates after 2 n - 2 units of time, if every message is transmitted in one unit of time, i.e., DDFS tree is constructed in 2 n - 2 units of time. For the completion of algorithm MCT, the CONNECT message has to traverse (n - 1) tree links which takes (n - 1)

units of time. This proves (b).□

Theorem 4.2.2.2:

Algorithm MCT uses exactly 3 (n - 1) messages and total number of bits used in communication is ≤ 3n (n - 1).

Proof:

It is shown in theorem 3.4.2 in section 3.4 that the algorithm DDFS uses exactly 2(n - 1) messages. It is clear from algorithm description in section 4.2 that MCT needs the CONNECT message to be transmitted after DDFS tree is determined. Since the tree is now available, the CONNECT message gets transmitted over the (n -1) tree links. This amounts to a total of 3(n - 1) messages for MCT.

Size of the DISCOVER message is at most n bits and that of CONNECT is n bits. Hence the communication cost of algorithm MCT bounded by 3n (n -1) bits. □

4.2.3. Discussion

We have presented and validated an algorithm for finding connectivity in communication networks. The algorithm is shown to have communication cost bounded by 3n(n-1) bits. In other words, the bit-wise message complexity of algorithm MCT is O(n^2). The earlier algorithms of Segall [48] (or adaptations of algorithms in [45]) have message complexity of 2ne $\log_2 n$ bits which in the worst case is of order n^3. In fact, it could be seen that the algorithm MCT has a lower communication cost compared to CT-5 of [48] whenever $e > 1.5 \frac{(n-1)}{\log_2 n}$.

4.3 Network Topology

In certain applications, a node may need to know the global topology information of the network such as number of participating nodes, their identity and also the global structure of the network. Global structure of a network is generally not maintained at every node since the network topology might change. The problem of learning network topology is defined as follows. For a communication network, a node i is said to have learned the network if it has learned the graph G(V,E) underlying the network, for all node j connected in the network.

The algorithm in [48] for finding network topology is as follows. Each node sends a message *Identity* which contains the adjacency list of the node to its neighbors. A node j which receives the *Identity* message records the received adjacency list and re-transmits *Identity* message to all its neighbors and then sends its adjacency list to its neighbors. The algorithm terminates when each node has received (and transmitted) the message from (to) all the nodes, the existence of which it has come to know from the neighbors. The algorithm may be started by one or more nodes. Slight modifications in the algorithms for CT by Segall [48] enable the use of CT algorithms for learning the network. The complexity of the network learning algorithm in [45] is shown to be O(ne) messages where each message size is of O(n) bits since the message carries the adjacency list of each node sending the message.

We present an algorithm MLT to learn the network topology. The algorithm MLT is based on the fundamental algorithm DDFS. MLT works in two stages.

Algorithm MLT:

1. In the first phase, we construct the DFS tree in a distributed fashion using DDFS.
2. The second phase begins with the root node sending a message called ADJACENT to its neighbors in DFS. The message mainly contains the adjacency list of the root node. Each node upon receiving the message then sends its adjacency list to all its neighbors in the DFS tree and re-transmits the received message to all its neighbors other than the node from where the message was received. The algorithm terminates when a node has received the adjacency lists of all nodes connected in the network. The total number of nodes connected in the network is learned by all nodes since the root node incorporates this additional information in the message ADJACENT it sends.

Algorithm DDFS could be modified in a number of ways. One such possibility is once a leaf node is discovered, it can send ADJACENT message to its father which in turn sends its adjacency list to the leaf node and re-transmits the received adjacency list to all other neighboring nodes in DFS as and when the tree arcs are discovered. However, for clarity we assume that the second phase of MLT begins only after DDFS is completed. Formal description of the algorithm MLT is given in appendix A4.3.

4.3.1. Complexity Analysis

Theorem 4.3.1.1:

Suppose node s receives the START . Then

(a). Algorithm MLT correctly determines the graph G underlying the network containing the node s.

(b). All nodes learn the graph G containing node s in finite time.

Proof:

The proof for (a) is as follows. Algorithm DDFS is shown to construct DFS correctly in section 3.4. The second stage of algorithm MLT begins with the root sending its adjacency list to all its children nodes in DFS. The algorithm MLT terminates when a node has received the adjacency lists of all nodes connected in the network. The graph of the network is now represented by the adjacency lists of all nodes that are connected in the network which uniquely determines the graph and hence the network structure.

Part(b) of the theorem is proved as follows. From theorem 3.4.3, DDFS uses 2n - 2 units of time for constructing DDFS tree, if every message is transmitted in unit time. For MLT to be completed, the adjacency list of each node has to be received by each other node. The ADJACENCY message of each node then has to traverse over (n -1) tree links and there are n such messages. Hence the maximum time for a node to receive adjacency lists from all other nodes connected in the network is at most n(n-1). This proves (b). □

Theorem 4.3.1.2:

Communication complexity of algorithm MLT is O(n^2) and uses no more than (n^2 - n - 2) messages.

Proof:

It is clear from the algorithm that the size of both DISCOVER and ADJACENT (or RADJACENT) messages is of O(n) bits. It is proved in theorem 3.4.3 that algorithm DDFS uses 2(n - 1) messages for constructing DDFS. In the second stage of algorithm MLT each node sends ADJACENT (RADJACENT in case of root node) over each of the (n -1) tree links. Since there are n nodes sending at most (n -1) messages the communication complexity of MLT is O(n^2).

The total number of ADJACENT (including RADJACENT) messages is at most n(n -1). Total number of messages used in MLT then is 2(n - 1) + n(n - 1), i.e., (n^2 - n - 1). □

4.3.2 Remark

We have presented and validated an algorithm for finding the structure of a communication network. Each of the messages namely DISCOVER, ADJACENT, and RADJACENT has a message size of O(n) bits. Therefore the bit-wise message complexity of the proposed algorithm MLT is O(n^3). As mentioned earlier, the algorithm presented in [45] has a worst case message complexity of O(n^3). Each message has O(n) bits and hence the bit-wise message complexity of

algorithm in [45] , in the worst case would be $O(n^4)$.

4.4 Biconnected Components

The problem of finding biconnected components is important in distributed systems, from the point of view of studying system reliability. In this section, we briefly discus the design of distributed algorithms for finding biconnected components in communication networks.

Definition 1: An articulation point (cut vertex) of a connected graph G is a vertex $v \in V(G)$, such that G-v has more than one component, i.e., the graph G is disconnected with the removal of v.

Definition 2: A graph G is biconnected if and only if it has no articulation points.

Presence of nodes that form articulation points indicate the vulnerability of the network. Sequential algorithms for finding biconnected components have been studied extensively in literature [58, 1]. These algorithms essentially employ the DFS algorithm and use DFS numbering. Chang [12] employs this technique to find biconnected components of asynchronous communication networks. The algorithm is shown to terminate in 2d(G) units of time using $O(n^3)$ messages. In [27] , a distributed algorithm for finding biconnected components is described which is based on defining new rules for detecting articulation points. The algorithm is shown to have time complexity of O(nk) and uses $O(n^2k)$ messages where k is the maximum node degree.

In section 3.4, we described a distributed depth first search algorithm which is similar to the sequential DFS algorithm. Now, using this algorithm, we can simulate Tarjan's algorithm [58] on a distributed system. This algorithm will then have a time and message complexities of O(nk) and O(e), respectively. Since the sequential algorithm is well known, the distributed algorithm is not described in detail.

CHAPTER 5

NETWORK LOCATION

5.1 Introduction

Network location problems deal with the task of selecting a site (node) subject to some optimality criterion. In general, problems of this nature are called facility location problems and they involve a wide range of application areas. Consider the following examples.

Example EX1: Suppose there is a plan to establish an emergency facility such as a hospital or fire station in a region consisting of several towns. The criterion for a problem like this is to minimize the response time between the facility and the location of possible emergency.

Example EX2: A facility such as a post office or power station has to be located in a region. The criterion to locate the facility is to minimize the total travel time for all the people in the region.

Example EX3: When constructing a superhighway, the problem of interest is the location of exits on the highway. The criterion to locate the exits is to make it convenient to the largest number of people.

In location problems, a fundamental issue is the selection of a suitable optimality criterion. In EX1 above, the idea is to minimize the maximum distance of travel from the facility. Such a criterion is called *minimax*. Locations that optimize the minimax criterion are referred to as the *centers* when the

problem refers to a network. The objective is to minimize the worst possible behavior. The other criterion is to optimize the average behavior. In the example EX2, the objective is to locate the post office in a place such that the average distance traveled by all people to the facility is a minimum. In EX1, if the criterion is to minimize the travel items to the facility rather than the response time, then again, we are looking for average behavior. This type of criteria is called the *minisum* and locations that optimize minisum criteria in networks are called *medians*. The third example EX3 defines locations called path centers.

In distributed systems, location of central facilities such as centers and medians play an important role in the design of efficient algorithms. Consider the following problem: a processor i in a distributed system contains a subset S_i of a universal set U. Determine the intersection of these subsets. This problem has been studied by Ramarao et.al. [43]. The algorithm runs in several iterations. In each iteration, every node in the network sends its data to a central node, to reduce both time and message complexities. It is shown that the algorithm is optimal. Similarly, the problem of distributed sorting is solved efficiently using central nodes [63]. Center based routing is studied by Owicki and others. In the case of broadcast, it is time efficient to use center based broadcast.

The concept of centrality is well studied in the field of graph theory with an significant contribution by Slater [9, 7, 23, 52, 53, 54]. Location problems applied to networks incorporating generalized centers and medians problems are dealt by Handler and Mirchandani [6]. Kariv and Hakimi present algorithmic approach to the location of p-centers and p-medians in networks in their excellent papers

[32, 33].

In this chapter, we present distributed algorithms for finding centers and medians of networks. Korach et. al. [34] presented the first distributed algorithms for this problem and developed unified algorithms for determining centers and medians of networks. However, we show a counter example for their algorithm and demonstrate that their algorithm fails to terminate for certain tree structures. We present both centralized and decentralized unified algorithms and ours is the only decentralized algorithm in literature. These results are extended to weighted tree networks. An interesting result on medians in weighted trees is presented. Finally, we generalize the algorithm for arbitrary networks, demonstrating the need for heuristic approaches.

5.2 Basic Concepts

In this section, we present basic graph theoretic concepts necessary to solve the problem of network location. Most of these results are used in the algorithm termination and computations. We introduce certain terminologies and define the notations used.

Definitions:

The distance d(u,v) denotes the length of the shortest path between nodes u and v. d(u,u) is assumed to be 0. A path between nodes u and v is denoted <u,v>.

The eccentricity $e(v)$ of a node v in G(V,E) is the max d(u,v) for all u$\in$ V(G).

$$i.e. \quad e(v) = \max_{u \in V(G)} d(u,v).$$

The radius r(G) is the minimum eccentricity of all the vertices.

$$i.e. \quad r(G) = \min_{u \in V(G)} e(u).$$

The diameter d(G) is the maximum eccentricity of the vertices.

$$i.e. \quad d(G) = \max_{u \in V(G)} e(u).$$

A node v is a **center** if e(v) = r(G) and the center of G is the set of all central nodes.

Define s(v) as the sum of the distances from node v to each other node in G.

$$i.e. \quad s(v) = \sum_{k \in V(G)} d(v,k)$$

A node v is a **median** if $s(v) = \min_{u \in V(G)} s(u)$ and the medians of G is the set of all median nodes.

A *Branch at a node u* of a tree T is a maximal subtree containing u as an end-point, i.e. number of branches at u is deg(u).

Weight W_u at a node u of a tree T is the maximum number of edges in any branch at u.

A node v is a **centroid** of T if W_u is minimum.

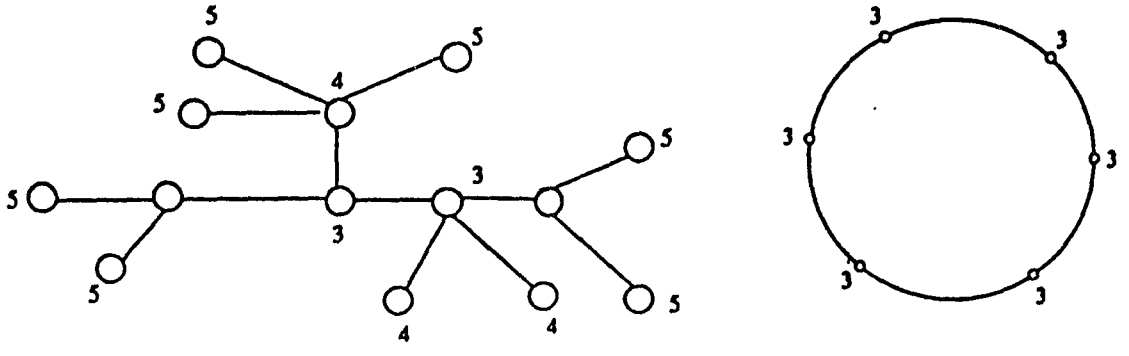

Figure 5.2.1 Centers: a. Tree and its eccentricities b. Self-centered graph

The concept of centers is illustrated in figure 5.2.1 where each node is marked with eccentricities. Medians and centroids are shown in figure 5.2.2 and 5.2.3 respectively.

The following notations have been used with respect to a tree T(V,E). Let v∈ V(T). N(v) denotes the set of neighbors of node v. T(v) represents the tree rooted at node v and T(i,v) represents the subtree rooted at v, containing the node i.

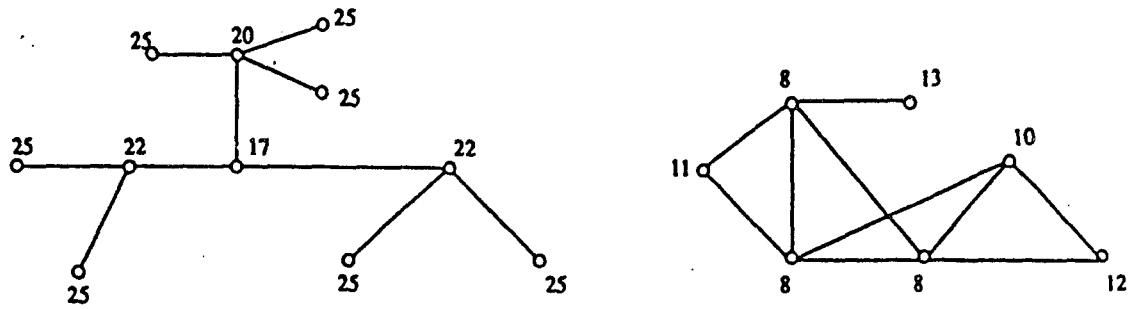

Figure 5.2.2 Medians: a. Tree and its s(v) b. Graph with multiple medians

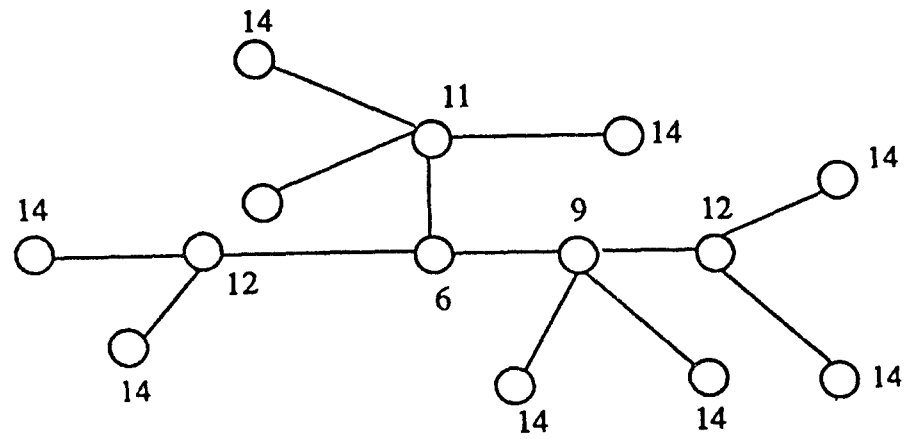

Figure 5.2.3. Centroids: Tree and weights w(v)

We now present some results related to centrality of graphs well known in graph theory. Proofs are omitted and may be found in [9, 23].

The basic result concerning centers is the classical theorem of Jordan [31]. For completeness, we state the theorem.

Theorem 5.2.1: Every tree has a center consisting of either one node or two adjacent nodes. □

The generalization of the above theorem, due to Harary and Norman [24], is of great importance in research on centrality.

Theorem 5.2.2: The center c(G) of any connected graph G lies within a block of G. □

Parallel to the above theorem, a similar result is shown true for medians of graphs by Truszczynski [60].

Theorem 5.2.3: The median M(G) of any connected graph G lies within a block of G. □

Theorem 5.2.4: The median of a tree consists of either a single node or a pair of adjacent nodes. □

Zelinka [64] showed that the studying centroid of a tree is identical to studying the medians of a tree.

Theorem 5.2.5 : Node v is a centroid node of a tree T if and only if it is a median node. □

5.3 Previous Results

In this section, we briefly discuss the algorithm of Korach, Rotem and Santoro [34] and present corrections to their algorithms. In their paper, Korach et.al. describe a basic algorithm to find the centers and medians for synchronous tree networks. It is shown that the same algorithm could be used for asynchronous tree networks and extended to general synchronous networks also. The basic algorithm functions as follows. A node i could be in one of the possible four states, viz. INACTIVE, AVAILABLE, ACTIVE, and SATURATED. during the execution of the algorithm. Before the start of the algorithm, all nodes are assumed to be in INACTIVE state. It is assumed that one node starts the algorithm.

Upon receiving a FORWARD message from a neighbor, a node in INACTIVE state transits to the AVAILABLE state if it is an internal node of the tree; to ACTIVE state if a leaf. An AVAILABLE node then sends the FORWARD message to all its neighbors except the node from where it received the FORWARD message. A leaf node returns BACKWARD message upon receiving a FORWARD message and changes to ACTIVE state. An AVAILABLE node changes to SATURATED state if it has received the BACKWARD message from all its neighbors. When a leaf node is the initiator, it sets itself to the ACTIVE state and sends a BACKWARD message to its neighbor. If a node in INACTIVE state receives the BACKWARD message, then it transits to ACTIVE state and propagates the BACKWARD message if it has only two neighbors; otherwise, it changes to AVAILABLE state and sends a FORWARD message to

its neighbors except the one from which the BACKWARD message was received. It is proved that one or two nodes reach SATURATED state in finite time and that every SATURATED node lies on the diameter path for synchronous tree networks.

We now present some tree structures for which the above algorithms do not find a correct solution. Also, it is later shown that the algorithm does not terminate for these structures. We first consider the algorithm for finding tree centers. Similar arguments could be extended to show that the algorithm fails to find medians also, since median finding is also based on the basic protocols.

The algorithm for finding tree centers (given in appendix) checks for the center whenever a node transits to the SATURATED state. Consider the tree represented by the connected graph P_n, $n \geq 2$, which has only a path , where n is the number of vertices in the graph.

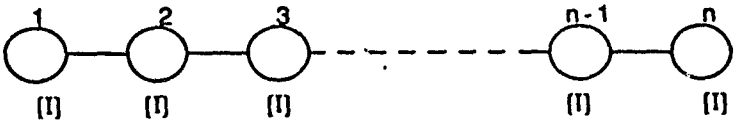

Figure 5.3.1 a. All nodes in IDLE state and node 1 initiates the algorithm.

See figure 5.3.1a. Each node is initially in the INACTIVE state indicated by "[I]" below each vertex in the figure. The vertices are numbered from 1 to n for convenience. Suppose v_1 starts the algorithm. Since v_1 is a leaf in P_n, it sends a BACKWARD message to its neighbor v_2 and transits to ACTIVE state. The node v_2 propagates the BACKWARD message to v_3 and changes to ACTIVE state. Similar action is taken by all nodes v_i, $1 \leq i \leq n\text{-}1$.

This is illustrated in figure 5.3.1b. When the node v_n receives the BACKWARD message, it changes its state to SATURATED.

According to the algorithm, the node v_n now calls the procedure RESOLVE (see appendix in [34]) to determine the center. Accordingly, v_n sends the BACKWARD message to v_{n-1}. Here is the problem with the algorithm. The node v_{n-1} is in ACTIVE state and according to the algorithm, there is no action defined for a node in ACTIVE state. A node in ACTIVE state does not react to any message and hence the algorithm fails to terminate.

The algorithm works correctly if all the internal nodes in the diameter path are in AVAILABLE state (except those in SATURATED state) since each such node sends a BACKWARD message upon receiving a BACKWARD message from all its neighbors. Apparently, it was assumed that only those nodes in AVAILABLE state lie on the diameter path. Figure 5.3.2 shows other tree structures for which the algorithm fails.

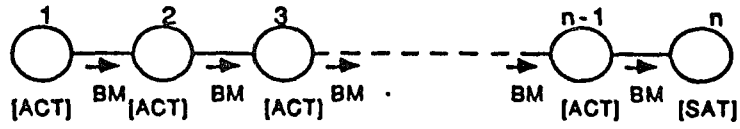

Figure 5.3.1 b. Node n in SATURATED state. (BM = Backward Message)
(ACT = ACTIVE state)

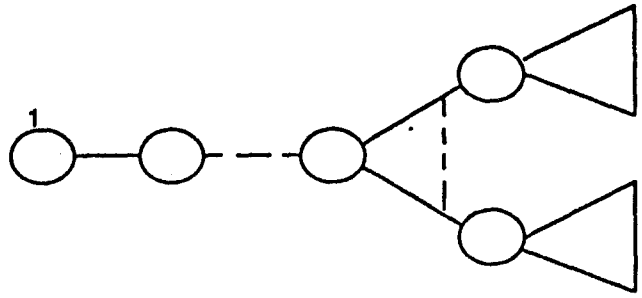

Figure 5.3.2 Node 1 is the initiator

Corrections to the algorithm

It is clear from the above discussion that for the algorithm to function correctly, a node in ACTIVE state should process the BACKWARD message, if received. It is clear from the algorithm that an ACTIVE node does not receive any other message. We need to include the following to the basic protocols.

If an ACTIVE node receives a BACKWARD message, then it behaves the same way as an AVAILABLE node does. It transits to SATURATED state and performs all the actions that are performed by an AVAILABLE node upon receiving BACKWARD messages from all the neighbors. Similarly, we need to add code for a node in ACTIVE state in the pseudo-code for finding center in [34] (in page 398). An ACTIVE node calls RESOLVE upon receiving BACKWARD message from all its neighbor and transits to SATURATED state. Similar modifications are required for median finding algorithm also.

5.4 Basic Centralized Algorithm and Applications

We now present the basic algorithm which is centralized. We study the application of this algorithm in finding centers and medians of asynchronous tree networks. The algorithm is called SDCM.

5.4.1 Basic algorithm SDCM

We present the basic algorithm used for locating center and median of asynchronous tree networks, when a single node starts the algorithm. The algorithms for center and median finding using SDCM are optimal within a constant, similar

to the worst case complexities of the algorithms in [7] for asynchronous tree networks. Formal description of the algorithm SDCM and its applications are given in appendix A5.4.1.

We employ two messages *viz* the EXPLORE and RETURN. The Explore message consists of only the message header where as the RETURN message consists of the message header and one or two data fields p_1 and p_2, depending on the application of the algorithm.

Let T(V(T),E(T)) be the tree of order n (i.e. |V(T)| = n). Suppose a node s $\in$ V(T), starts the algorithm. The tree T can now be viewed as a tree T(s) rooted at s, with subtrees T(1,s), T(2,s),.., T(k,s), $1 \leq k \leq n-1$; where nodes 1,2,..,k $\in$ N(s)., i.e., T(s) = { T(1,s), T(2,s),.., T(k,s) }. Clearly, $V(T(i,s)) \cap V(T(j,s)) = \{s\}$, $i \neq j$. During the execution of the algorithm, each node in T may be in one of the following states: { IDLE, WAIT, TERMINAL } and initially, all nodes are assumed to be in IDLE state. The algorithm SDCM is as follows.

The start node *s* broadcasts the EXPLORE message to all its neighbors and transits to the WAIT state. If the start node is a leaf, then it behaves as if it received the EXPLORE message and sends the RETURN message to its neighbor and enters WAIT state.

1. When an IDLE node i receives EXPLORE message from a node j $\in$ N(i);

    1.1 if i is a leaf, then it changes state to WAIT and sends the RETURN message to j.

1.2 otherwise, it broadcasts EXPLORE message to all $k \in N(i)-\{j\}$ and transits to the WAIT state.

2. When a node i in IDLE state receives the RETURN message from a neighbor j;

{ this occurs when a leaf node starts the algorithm }

2.1 if $|N(i)| = 1$, i.e., i is a leaf node, then it transits to the TERMINAL state and the algorithm stops.

2.2 if $|N(i)|=2$, then the node i transits to the WAIT state and sends the RETURN message to the other neighbor ($\neq j$).

2.3 otherwise, it sends EXPLORE message to all its neighbors except j and transits to the WAIT state.

3. When a node i in WAIT state receives RETURN message from $j \in N(i)$;

3.1 it computes the various parameters depending upon the application and

3.2 if it has received RETURN message from all its neighbors except the one from where it had received EXPLORE message, then it sends RETURN message to this neighbor.

3.3 if the total received RETURN message = $|N(i)|$, then the node changes to the TERMINAL state and the algorithm stops.

We now describe a few important properties of the algorithm SDCM which will be required while proving the properties of the algorithms for the applications. Discussions on the complexity analysis and proof of correctness of the algorithm are provided under each application in the subsequent sections.

Suppose a node $s \in V(T)$ starts the algorithm .

Lemma 5.4.1.1: A node in WAIT state does not receive the EXPLORE message from any of its neighbors.

Proof:

Suppose a node i in WAIT state receives a EXPLORE message from its neighbor j; $i,j \in V(T)$, $i \neq j$. Let k be the node that sent EXPLORE message to an IDLE node i, $k \neq j$. Since s is the only initiator of the algorithm there exist two paths from s to i; one path via j and the other via k. A contradiction to the assumption that T is a tree. □

Lemma 5.4.1.2: When the algorithm SDCM stops, there must be a node in $i \in V(T)$ which reaches the TERMINAL state in finite time.

Proof: The EXPLORE message initiated at some start node s traverses down all the subtrees rooted at s, to each leaf node in the tree. Each leaf node responds to the EXPLORE message with the RETURN message which is echoed by all internal nodes until it reaches the start node as given in the algorithm.

If the start node is a leaf node, then we have two cases; $T = P_n$, the path graph and $T \neq P_n$. When $T = P_n$, the other leaf reaches the TERMINAL state as it receives the propagated RETURN message. When $T \neq P_n$, the node k nearest

the start node s such that deg(k) > 2, receives the RETURN message propagated from the start node and thereafter, acts as the start node. Eventually, node k reaches the TERMINAL state in finite time. □

In the subsequent sections, we show the applications of the algorithm SDCM in determining the centers and medians of asynchronous tree networks. Note that the center and median finding differ mainly in the computations performed at each node and in the algorithm termination.

5.4.2 Center finding algorithm (SCEN):

We now describe some important properties of trees and state some important results concerning centers of trees. Proofs and other details of these properties can be found in [23].

P5.4.2.1: Every diameter path contains all centers of a tree.

P5.4.2.2: For a tree, $r(T) = \lceil \frac{d(T)}{2} \rceil$.

We present the extensions for algorithm SDCM to determine the centers of the tree network.

The RETURN message consists of one parameter p_1. Each leaf sends RETURN(p_1) with $p_1 = 1$. A node i that receives the RETURN(p_1) message performs the following computations:

it computes h_1 and h_2 as the two largest p_1's received from neighbors i and j with $h_1 \geq h_2$. (initially, both h_1 and h_2 are set to 0.)

A node sends the RETURN(p_1) message with $p_1 = h_1 + 1$, thus computed at that node.

A node i in TERMINAL state performs the following: Let h_1 and h_2 be received from neighbors j and k respectively.

1. if $h_1 = h_2$, then the node i is the center and the algorithm terminates.
2. if $h_1 - h_2 = 1$, the node i becomes the center and sends RETURN message to node j. (Actually, both nodes i and j are centers.)
3. otherwise, $h_1 > h_2 + 1$; node i sends RETURN($h_2 + 1$) to node j. Clearly, node j is in WAIT state. Upon receiving the RETURN message from i, it reaches the TERMINAL state and performs similar actions until the centers are determined.

Lemma 5.4.2.1 Let node i be a node in TERMINAL state and h_1= h((T(j, i)), h_2= h(T(k, i)) be the two largest heights obtained at node i from neighbors j and k respectively. Then

(1) If $h_1 = h_2$, then node i is the only center of T;

(2) If $h_1 = h_2 + 1$, then node i and its neighbor node j, are the two centers of T.

Proof. Note that h_1 = h(T(i)) which is the height of the spanning tree T(i). Consider case (1), $h_1 = h_2$. We prove that any other node $x \in V(T)$, $h(T(x)) > h_1 = h(T(i))$. Let a, b be two leaf nodes in T(j, i), T(k, i) respectively, such that d(a, i) = d(i, b) = h_1. For any node $x \in V(T)$, $x \neq i$, we have either $x \notin V(T(j, i))$ or $x \notin V(T(k, i))$. Without loss of generality, assume $x \notin V(T(j, i))$. Then the only path

from x to node a must pass node i, hence d(x, a) = d(x, i) + d(i, a) > h_1. That is, h(T(x)) > h_1, hence node i is the only center.

Consider case (2). First, we note that h(T(i)) = h(T(j)) = h_1. Let nodes a, b be two leaf nodes in T(j, i) and T(i, j) respectively such that d(a, i) = d(j, b) = h_1. By similar argument, any node x other than i and j, we have either x $\notin$ V(T(j, i)) or x $\notin$ V(T(i, j)), and so h(T(x)) > h_1 + 1. □

Lemma 5.4.2.2 Let node i,j,k be as in lemma 5.4.2.1. If $h_1 > h_2 + 1$, then i is not a center, and the center of T lies in T(j, i) - {i}. Moreover, the height of the sub-tree T(i, j), which is rooted at node j and contains node i, is h_2 + 1.

Proof. If node i is a leaf node, the conclusion is trivially true . Otherwise, the spanning tree T(i) has more than one subtrees T(n1, i), T(n2, i), ..., T(nr, i), where n1, n2, ..., nr are all the neighbor nodes of node i, r ≥ 2. Let a, b be two leaf nodes in T(j, i) and T(k, i) respectively such that d(a, i) = d(a, j) +1 = h_1 and d(i, b) = d(j, b) -1 = h_2. For any node x $\notin$ V(T(j, i)), we have d(x, a) = d(x, i) + d(i, a) > h_1, hence h(T(x)) > h(T(i)) = h_1, node x is not a center. So the center(s) must be in the subtree T(j, i). Also we notice that the height of T(i, j) = 1 + T(k, i) = h_2 + 1. The longest path in T starting at node j will be ended in either T(j, i) or T(i, j); so h(T(j)) = max {d(a, j), d(j, b)} = max {h_1 -1, h_2 +1} < h_1, hence node i is not a center. □

Lemma 5.4.2.3 Algorithm SCEN correctly finds the center(s) of the tree T.

Proof. By Lemma 5.4.2.1, a node i will receive RETURN message from all its neighbors and hence will reach the TERMINAL state. By Lemma 5.4.2.1, we

know that the algorithm will correctly find the center(s) of T, if $h_1 - h_2 \leq 1$. In case $h_1 - h_2 > 1$, by Lemma 5.4.2.2, we know the center(s) of T lies in sub-tree T(j, i) - {i}, in the algorithm, node i will send RETURN($h_2 + 1$) to node j, which is a possible candidate for a center of T. Node j now having received RETURN from all of the neighbors transits to TERMINAL state and the difference between the two largest heights of sub-trees rooted at j will be $|(h_1 - 1) - (h_2 + 1)| = |h_1 - h_2 - 2| < |h_1 - h_2|$. Hence, such a process must terminate in finite steps when $h_1 = h_2$ or $h_1 = h_2 + 1$. Then by Lemma 5.4.2.1 and 5.4.2.2, the center(s) of T is found. □

Theorem 5.4.2.1 Algorithm SCEN needs at most $2\, h_{max}$ + d(T)/2 time units.

Proof. The propagating of EXPLORE message from the start node s and the collecting RETURN message to start node s requires at most $2\, h_{max}$ time units. After that, we may need to move to a center c ($c \neq s$) by sending RETURN message along path <s, c>, which needs additional d(s, c) time units. Since node c is a center of T, so d (s, c) ≤ h(T(c)) = d(T)/2. The total time for algorithm Tree-center is therefore bounded up by $2\, h_{max}$ + d(T)/2. □

Theorem 5.4.2.2 Algorithm SCEN determines the centers using at most 2(n − 1) + d(T)/2 messages.

Proof. Suppose node j is in TERMINAL state. At the time the node j received RETURN from all nodes in N(j), at most two messages have been passed on each link; one is EXPLORE the other is RETURN. The total number of messages is 2(n − 1). Then in moving from node j to a center c, we may need to

pass RETURN along path <j, c>, so the number of messages for this process is at most h(T(c)) = d(T)/2, by the property of trees.

5.4.3 Median finding algorithm (SMED):

We first discuss the some important properties of medians of trees. The following properties of trees are useful in designing algorithms for finding medians. All these properties have been proved in [34] and we state them here for later use.

P5.4.1: For v∈ V(T), let s(v) = $\sum_{u \in V(T)} d(v,u)$. Then, for uv ∈ E(T),

$$s(u)=s(v)+|V(T(v,u))|-|V(T(u,v))|.$$

P5.4.2: For uv ∈ E(T), let Δ (u,v) = s(u) - s(v). Then

$$\Delta(u,v) = |V(T)| + 2 - 2|V(T(u,v))| = -\Delta(v,u).$$

P5.4.3: A node u∈ V(T) is a median of T if and only if, for all v ∈ N(u), Δ(v,u) ≥ 0.

It is clear from the definition of median and properties of trees described above that finding median is similar to the process of finding centers except that the computations performed at each node and the termination predicates are different. We now describe the extensions of the algorithm SDCM, and computations performed at each node for determining the medians.

The RETURN message consists of two parameters p_1 and p_2. The parameter p_1 contains the sum of distances and p_2 contains the number of nodes in the subtree rooted at a node. A node i sends RETURN(p_1, p_2) to node j with

$p_1 = \sum_{k \in V(T(i,j))-\{j\}} d(i,k)$ and $p_2 = |V(T(i,j))| - 1$.

Each leaf node sends RETURN(p_1, p_2) with $p_1 = 0$ and $p_2 = 1$. A node i that receives RETURN(p_1,p_2) performs the following computations.

i. lsum = lsum + p_1 +p_2 { sum of distances received so far; initially lsum is 0 }

ii. lnum = lnum + p_2 { num of nodes in the subtrees known so far; initially 0 }

iii. determines the neighbor k such that

|V(T(k,i))| > |V(T(l,i))| for all l∈ N(i)-{k}, from which the RETURN message has been received; and computes:

stnum = |V(T(k,i))| - 1 { p_2 in the RETURN message from node k }

stsum = $\sum_{x \in V(T(k,i))-\{i\}} d(k,x)$ { p_1 in the RETURN message from node k}

stnode = k.{ neighbor with largest subtree }

When a node i sends the RETURN message, the parameters are set as: p_1 = lsum, and p_2 = lnum + 1.

A node i in TERMINAL state starts locating the medians of the tree. It computes the following which are similar to the results stated as properties P5.4.1, 5.4.2 and 5.4.3 in the beginning of this section.

1. computes Δ = (lnum + 1) - 2 stnum.

2. if $\Delta \geq 0$ then, node i is the median. It sends RETURN(lsum-stsum-

stnum,lnum-stnum +1) to node *stnode*, to determine another median, and exits the algorithm.

3. otherwise, $\Delta < 0$, and hence the median lies along a path in T(*stnode*, i); node i sends RETURN(p_1,p_2) to node *stnode* with p_1 = lsum - (stnum + stsum), and p_2 = lnum - stnum + 1. t is clear that the *stnode* is in WAIT state. Upon receiving the RETURN message, it reaches TERMINAL state and performs the computations described above; thus propagating the information until medians are determined.

It is clear from the algorithm and from the properties of medians described that the algorithm SMED determines the medians correctly, in finite time. We now present the complexity measures.

Suppose a node $s \in V(T)$ starts the algorithm SMED and let $m \in V(T)$ be a median of T.

Theorem 5.4.3.1: The algorithm SMED determines the median m of a tree T in at most $2h_{max}$ + d(s,m) units of time, if all messages are delivered in one unit of time.

Proof: The first term in the time complexity measure above accounts for the time needed for the start node reaching the TERMINAL state, as shown in theorem 5.4.1.1. The second term viz. d(s,m) accounts for the time required to move from the start node to the median.

□

Theorem 5.4.3.2: The algorithm SMED determines a median using at most 2(n-

1) + d(s,m) messages.

Proof: Follows from the theorems 5.4.2.2 and 5.4.3.1.

□

Clearly, the algorithm uses at most 2 more messages to determine the other median, if it exists.

5.5 Basic algorithm MDCM and applications.

In this section, we consider the problem of finding centers and medians in asynchronous tree networks, when multiple nodes initiate the algorithm. All these algorithms follow a basic protocol which has been described as algorithm MDCM. Extensions of algorithm MDCM for applications in finding diameter, centers and medians are described in subsequent sections. As in the earlier section on algorithm SDCM, we examine certain properties of the basic algorithm MDCM and describe specific properties under each application. It is easy to see that the algorithms based on MDCM degenerate to the single node starting case discussed in section 5.4 when one node starts the algorithm, but this algorithm remains different from SDCM. Formal description of the algorithm MDCM and its applications are given in appendix 5.5.1.

5.5.1 Algorithm MDCM:

We employ two messages, the EXPLORE and the RETURN message which are similar to those in algorithm SDCM. A node could be in any one of the following states: $\{IDLE, ACTIVE, TERMINAL\}$ and initially all nodes are assumed

to be in the IDLE state. Suppose a set of nodes S = { $s_1, s_2,..,s_k$ }, S $\subset$ V(T), start the algorithm. The algorithm proceeds as follows.

Each start node s_i sends the EXPLORE message to all its neighbors. If s_i is a leaf node, then it acts as if it received an EXPLORE message and sends the RETURN message to its neighbor. After this, each start node changes to the ACTIVE state.

1. When a node i in IDLE state receives an EXPLORE message from node j $\in$ N(i);

   1.1 if i is leaf node, then it changes to ACTIVE state and sends the RETURN message to its neighbor.

   1.2 Otherwise, it changes to ACTIVE state and broadcasts the EXPLORE message to all nodes k $\in$ N(i)-{j}.

2. When a node i in IDLE state receives RETURN message;

   { can happen when some leaf node starts the algorithm }

   2.1 if it has received RETURN from all its neighbors except one, j,say, then it sends RETURN message to node j. The node transits to ACTIVE state.

   2.2 if it has received RETURN message from all its neighbors, then it transits to the TERMINAL state and the algorithm MDCM stops. { each application of algorithm MDCM starts from this point.}

   2.3 otherwise, it changes to ACTIVE state and broadcasts the EXPLORE

message to all its neighbors in N(i)-{j}.

3. When a node i in ACTIVE state receives an EXPLORE message, the message is ignored. { implies that some other node/s started in the subtree containing node i.}

4. When a node i in ACTIVE state receives a RETURN message from node j∈ N(i);

   4.1 if the RETURN message has been received from all its neighbors, then it changes to TERMINAL state and the algorithm MDCM stops.

   4.2 if the RETURN message is received from all its neighbors except one, k, say, then it sends RETURN message to the node k.

   4.3 otherwise, computes the parameters and remains in the same state.

Properties:

We present an important property of the algorithm MDCM. The complexity analysis and proof of correctness are discussed under each application in the subsequent sections.

Lemma 5.5.1:

Suppose a set of nodes S = $\{s_1, s_2,..,s_l\}$, S ⊆ V(T), start the algorithm. There will be exactly one node or two adjacent nodes that reach the TERMINAL state, in finite time.

Proof:

A node reaches TERMINAL state whenever it has received the RETURN message from all its neighbors. Also, a node sends RETURN message to its neighbor if and only if it has received RETURN from all its neighbors except one or if it is a leaf. So, if a node n_i receives RETURN from a neighbor n_j , then it is implied that the necessary computations are completed with the subtree rooted at n_j.

Suppose there are three nodes n_i, n_j and n_k, i≠j≠k, (not necessarily adjacent) that reach the TERMINAL state. This implies that RETURN message was sent by n_i to the subtrees that include n_j and n_k; and by n_j to n_i and n_k; and by n_k to n_i and n_j. A contradiction to the facts stated above.

It is clear that there could be one node that reaches the TERMINAL state. If two nodes n_i and n_j both reach the TERMINAL state, both of them must have sent the RETURN message to each other and hence must be adjacent. □

5.5.2 Center finding algorithm (MCEN):

The algorithm MCEN determines the center of the tree when multiple nodes start the algorithm. It is based on the basic algorithm MDCM.

The RETURN message consists of one parameter, p_1 (say). Each leaf node sends the RETURN message with $p_1 = 1$. Each node i ∈ V(T) that receives a RETURN(p_1) message performs the following computations.

i. computes h_1 and h_2 as the two largest p_1's received from neighbors j and k, with $h_1 \geq h_2$. (both h_1 and h_2 are initially set to 0.)

An internal node sends RETURN(p_1) with $p_1 = \max(h_1, h_2) + 1$.

A node i in TERMINAL state does the following. Let h_1 and h_2 be obtained from neighbors j and k, respectively.

1. if $h_1 = h_2$, then it changes its status to center and the algorithm terminates.

2. if $h_1 - h_2 = 1$, then it changes its status to center and sends RETURN($h_2 + 1$) to node j and exits the algorithm.

3. otherwise, $h_1 > h_2 + 1$, sends RETURN($h_2 + 1$) to node j. Clearly, node j is in ACTIVE or TERMINAL state. Upon receiving the RETURN message, if in ACTIVE state, node j transits to TERMINAL state as given in the basic algorithm and performs the above operations; thus propagating the RETURN message until the centers are determined.

When a node in TERMINAL state receives the RETURN message, the message is ignored.

We now present the analysis of the algorithm MCEN. It is clear that lemma 5.4.2.3 holds in case of MCEN also, by the properties of trees.

Lemma 5.5.2.1 The algorithm MCEN correctly determines the centers of the tree T and terminates in finite time.

Proof: It is shown in lemma 5.5.1.1 that one node or two adjacent nodes reach TERMINAL state in finite time. In the case when one node is in TERMINAL state, lemma follows from the lemma 5.4.2.3.

Let j and k be the two neighboring nodes in TERMINAL state. If h(T(j)) = h(T(k)), then it is exactly the case that $h_1 = h_2 + 1$, for both nodes j and k. Then by the lemma 5.4.2.1, nodes j and k will be identified by the algorithm as centers. Otherwise, without loss of generality, assume that h(T(j)) > h(T(k)); then from lemma 5.4.2.2, the centers lie in T(k,j)-{j}. According to the algorithm (step 3), node j sends RETURN($h_2 + 1$) to node k. The node k will determine the centers as in the algorithm SCEN, and therefore the centers are determined correctly in finite time, as shown in the lemma 5.4.2.3. □

Theorem 5.5.2.1: The algorithm MCEN finds the centers of a tree in at most 2 h_{max} + d(T)/2 units of time, if all messages are delivered in one unit of time.

Proof: Follows from theorem 5.4.2.1.

Theorem 5.5.2.2: Algorithm MCEN determines the centers of the tree using at most 3(n – 1) + d(T)/2 messages.

Proof: In the worst case, all nodes might start the algorithm , simultaneously and each will send the EXPLORE message to their neighbors, except the leaf nodes which send the RETURN messages. This amounts to 2(n – 1) messages. There will be at most (n – 1) RETURN messages before the node/s can reach the TERMINAL state. Now to move to the center c from a node i in TERMINAL state, it takes at most d(i,c) ≤ d(T)/2, by the property of the tree. Hence the message complexity of the algorithm MCEN is 3 (n - 1) + d(T)/2. □

5.5.3 Median finding algorithm (MMED):

The median finding algorithm MMED, when multiple nodes start the algorithm parallels that for single node starting case (SMED) and the center finding algorithms; and hence is not discussed in detail. However, the complexities are different since multiple nodes start the algorithm and any node in the tree could be in TERMINAL state at the end of the algorithm MDCM. We present the worst case complexities of the algorithm MMED.

Suppose a set of nodes $S = \{ s_1, s_2, .., s_k \}$, $S \subseteq V(T)$, $S \neq \emptyset$; start the algorithm MMED.

Theorem 5.5.3.1: The algorithm MMED determines the medians of a non-trivial tree T in less than $2h_{max} + d(T)$ units of time, if all messages are delivered in one unit of time.

Proof: The first term follows from theorem 5.5.2.1. Suppose a node i is in TERMINAL state and node m is a median. Then, $d(i,m) < d(T)$ since no leaf could be a median in a non-trivial tree, and hence locating a median starting from the TERMINAL node needs $< d(T)$ units of time.

□

Theorem 5.5.3.2: The algorithm MMED determines the medians of a tree T using less than $3(n-1) + d(T)$ messages.

Proof: The first term in the message complexity follows from theorem 5.5.2.2. The second term follows from theorem 5.5.3.1.

□

5.6 Algorithms for Weighted Tree Networks:

Consider an asynchronous tree network representing a communication network. As stated in the earlier section, it is realistic to represent the network as a weighted network; the weights on links indicating cost of communication, instead of assuming links with constant communication costs. The cost of communication between two nodes in the network could be a function of multiple parameters such as the distance between the two nodes, link traffic, link reliability etc. In this section, we consider the problem of finding centers, diameter, and medians of such asynchronous weighted tree networks.

Let T(V,E,f) represent an asynchronous weighted tree where V denotes the set of vertices in T, E represents the set of edges in the tree and f denotes the weight function which maps f: E(T) $\rightarrow R^+$. We write w(i,j) to denote the communication cost for delivering a message from node i to node j (i.e. over the link (i,j)). We present the modifications for the algorithms SDCM and MDCM to determine the various topological parameters. We use the following notations in the subsequent sections.

d_w(u,v) = shortest weighted distance from node u to node v.

5.6.1 Center finding algorithm:

We now present modifications to the algorithms SCEN and MCEN for

determining the centers of a weighted tree. The basic protocols remain the same as that for the unweighted trees, but the computations performed at each node are different. The modifications are as follows.

A leaf node i sends RETURN(p_1) to its neighbor j, with p_1 = w(i,j).

A node i that receives the RETURN(p_1) message performs the following computations.

i. computes h_1 and h_2 as the two largest p_1's received from neighbors i and j with $h_1 \geq h_2$.(h_1 and h_2 are initially set to 0 .)

When a node i sends a RETURN(p_1) message to node j, the parameter is set as $p_1 = h_1 + w(i,j)$.

We now consider the problem of determining the centers after a node reaches the TERMINAL state. For node j, $k \in V(T)$, $j \neq k$, let h_1 and h_2 computed at node i be the heights of the subtrees T(j,i) and T(k,i). A node i in TERMINAL state performs the following operations.

i. if $0 \leq h_1 - h_2 < w(i,j)$, the node i is the center and the algorithm terminates.

ii. if $h_1 - h_2 = w(i,j)$, then node i becomes the center and sends RETURN(h_2 + w(i,j)) to node j. In this case, both the nodes i and j are centers.

iii. Otherwise, $h_1 > h_2 + w(i,j)$; node i sends RETURN(h_2 + w(i,j)) to node j. Clearly, node j is in WAIT (or ACTIVE in case of MDCM) or TERMINAL state. If j is in TERMINAL state, it ignores the RETURN message. Otherwise, it attains the TERMINAL state and performs similar operations for a node in TERMINAL state.

Lemma 5.6.1.1:

Let node i be in TERMINAL state and h_1 = h(T(j,i)) and h_2 = h(T(k,i)) be the two largest heights of subtrees rooted at node i. Then,

i. if $0 \leq h_1 - h_2 <$ w(i,j), then node i is the only center of T.

ii. if $h_1 - h_2$ = w(i,j), then both node i and node j are the centers of T.

iii. otherwise, $h_1 > h_2$ + w(i,j); node i is not the center and the center lies in subtree T(j,i) - {i}.

Proof:

1. When $h_1 = h_2$, the result follows from lemma 4.3.1. Consider the case when $0 < h_1 - h_2 <$ w(i,j). We prove that for any vertex x $\in$ V(T), x$\neq$ i, h(T(x)) > h_1 = h(T(i)).

Let a and b be two leaf nodes in T(j,i) and T(k,i) respectively, such that d_w (a,i) = h_1, and d_w(b,i) = h_2. For a node x $\in$ V(T), x$\neq$ i, we have x $\in$ V(T(j,i)) or x $\in$ V(T(k,i)). Without loss of generality, assume x $\in$ V(T(k,i)). Then the only path from x to the leaf node a must pass through node i; hence d_w(x,a) = d_w(x,i) + d_w(i,a) > h_1. Also note that node j is not a center since

$$\begin{aligned} d_w(j,b) &= h_2 + w(i,j) \\ &> h_1 - w(i,j) + w(i,j) \\ &> h_1. \end{aligned}$$

2. When $h_1 = h_2$, the result follows from case 2 in lemma 5.4.2.1, considering weighted distances.

3. The case when $h_1 > h_2$ + w(i,j), is similar to that in lemma 5.4.2.2.,and the

proof follows from lemma 5.4.2.2, considering weighted distances.

Hence the modified algorithm determines the centers of a weighted correctly in finite time. The time and message complexities remain the same as that in SCEN and MCEN.

5.6.2 Median finding algorithm:

We present some basic properties of medians for weighted trees. We show an interesting fact that median location in a tree depends only on the structure of the tree and is independent of whether the tree is weighted or unweighted. However, the weight determines only the value of the medians. The properties discussed below could be found in [7] with respect to unit weight trees; we extend these results to a general weighted tree.

Let T be a weighted tree with w(i,j) representing the communication cost between nodes i and j. Let $w_i = \sum_{j=1}^{j=n} d_w(i,j)$. Also, w(i,i) = 0.

Lemma 5.6.2.1:

Let i,j be two distinct, adjacent nodes. Then, $w_i = w_j$ + w(i,j) [|V(T(j,i))| - |V(T(i,j))|].

Proof:

By definition of w_i, we have,

$$w_i = \sum_{k \in V(T)} d_w(i,k) = \sum_{k \in V(T(i,j))-\{j\}} d_w(i,k) + \sum_{k \in V(T(j,i))-\{i\}} d_w(i,k)$$

However, V(T) = V(T(i,j)) ∪ V(T(j,i)) - {i,j}.

$$d_{w(i,k)} = \begin{cases} d_w(j,k) + w(i,j) & \text{for } k \in V(T(j,i)) - \{i\} \\ d_w(j,k) - w(i,j) & \text{for } k \in V(T(i,j)) - \{j\} \end{cases}$$

$$Hence, w_i = \sum_{k \in V(T(i,j)) - \{j\}} d_w(j,k) + w(i,j)\,[\,|V(T(j,i))| - 1\,] + \sum_{k \in V(T(j,i)) - \{i\}} d_w(j,k) - w(i,j)\,[\,|V(T(i,j))| - 1\,]$$

$$= w_j + w(i,j)\,[\,|V(T(j,i))| - |V(T(i,j))|\,].$$

Hence, the lemma holds. □

We define Δ(i,j) = w_i - w_j. Then we have,

$$\Delta(i,j) = w(i,j)\,[\,|V(T(j,i))| - |V(T(i,j))|\,].$$

$$but,\ |V(T(i,j))| = |(V(T)| + 2 - |V(T(j,i))|.$$

$$\Delta(i,j) = w(i,j)\,[\,n + 2 - 2\,|V(T(i,j))|\,] \quad where\ n = |V(T)|.$$

Lemma 5.6.2.2: For a node i∈ V(T), there exists at most one node j ∈ N(i) such that Δ(j,i) ≤ 0.

Proof:

Suppose there exist nodes k,r∈ V(T), k≠ r such that Δ(k,i) = Δ(r,i) ≤ 0. Then by definition,

$$\Delta(k,i) = w(i,j)[\,n + 2 - 2|V(T(k,i))|\,] \le 0 \quad \text{and}$$

$$\Delta(r,i) = w(i,j)[\,n + 2 - 2|V(T(l,i))|\,] \le 0.$$

Since w(i,k) and w(i,r) are both > 0, it is sufficient to consider only the terms in brackets in the equations above. Adding these two terms, we have,

$$2(n+2) - 2(|(V(T(k,i))|) + 2(|V(T(r,i))|) \le 0$$

which is impossible since

$$|V(T(k,i))| + |V(T(r,i))| \leq n + 1.$$

□

It can now be easily shown that a node $i \in V(T)$ is a median if and only if for all $j \in N(i)$, $\Delta(j,i) \geq 0$, and that there exist at most two medians in a weighted tree network. The proofs for these statements are omitted in this paper since they are identical to those for unweighted trees and could be found in [7]. Thus, these results hold for weighted trees also, and hence the following theorem follows.

Theorem 5.6.2.1:

Median location in a tree is independent of whether the tree is weighted or unweighted. □

The algorithm for median finding is the same as that for unweighted trees discussed in the earlier sections except that weighted distances are used in the computations. It is clear that the complexities remain the same as that for unweighted trees.

In table 1, we summarize the performance of our algorithm and the algorithm of Korach et.al. [34] for asynchronous tree networks considering worst case time and message complexities. However, it should be noted that for the model assumed in [34] , the time complexity is not defined for the algorithms for asynchronous networks.

Table 5.1: Summary of performance

Algorithm	Korach,Rotem and Santoro (July 1984)		Sharma,Chen and Iyengar	
Single node start:	Yes		Yes	
Multiple node start:	No*		Yes	
Weighted trees;	No**		Yes	
Complexities	Time	Message	Time	Message
Single node start:				
a. center	Not defined	$\leq 2(n-1)+d(T)/2$	$\leq 2h_{max} + d(T)/2$	$\leq 2(n-1)+d(T)/2$
b. median	Not defined	$<2(n-1)+d(T)$	$<2h_{max} + d(T)$	$<2(n-1)+d(T)$
Multiple node start:				
a.center	-	-	$\leq 2h_{max}(S) + d(T)/2$	$\leq 3(n-1) + d(T)/2$
b.median	-	-	$<2h_{max}(S) + d(T)$	$< 3(n-1) + d(T)$

* Not direct; in the first phase, leader election need be done and then the leader acts as the start node. This will increase the time and message complexities of the algorithm.

** It has been mentioned that extensions to weighted trees are possible, but details are not included.

CHAPTER 6

NETWORK RECOGNITION

6.1 Introduction

The design of distributed algorithms for graph problems is crucial to the effective utilization of a distributed system. It has already been explained that in a distributed system, learning topological configuration is vital to the design and/or use of suitable fast algorithms to solve various computational problems on a distributed system. In chapter 5, we demonstrated the need for locating facilities such as centers and medians in networks in an effort towards obtaining topological information. In this chapter, we develop motivation for recognizing structural configuration of networks and design distributed algorithm for network recognition.

The technique of designing efficient algorithms by restricting graph structures is well known to the domains of sequential and parallel processing. There exist several problems in literature which when restricted to certain graph structures result in optimal or efficient solutions. For example, consider the problem of partitioning a vertex set V into disjoint sets V_1andV_2 such that the sum of weights of the edges from E that have one end point in V_1 and the other in V_2 is at least K where K is an integer. This problem is shown to be NP-complete in [22]. However, it is also shown that the problem has polynomial solution if the graph is planar. In [22] several problems that are shown to be NP-complete

have polynomial solution for certain graphs. Consider the problem of finding p-centers and p-medians of graphs. It is shown to be NP-complete for arbitrary graphs. However, there exist polynomial solution for trees. There are several such problems in literature of parallel algorithms that have feasible solutions for certain graph structures.

The observations made above hold true for distributed systems also. It is evident from chapter 5 that the problem of locating centers and medians has linear time and message complexities when we restrict the underlying network of distributed system to be a tree. It is however, not true when we consider arbitrary graphs. Specifically, the message complexity of the algorithm to locate centers and medians for arbitrary graphs increases to $O(n^3)$. It is shown in [19] that problems such as finding shortest path, routing, etc. become easy to handle when the graph is planar. In general, algorithms for general graphs are expensive in terms of both time and message complexities, where as algorithms for certain topologies are much faster and easy to design. This is the basic motivation behind designing distributed algorithms for network recognition problems.

Network recognition basically amounts to the task of performing preprocessing on a system. In the literature, most of the works consider the following topologies for algorithm design: trees, rings, star, mesh (grid), complete graphs, bipartite graphs, planar graphs, and general graphs. Problem of network recognition can now be stated as follows.

Problem: Given a graph G(V,E) underlying a communication network, determine if the topology of the network is one of the set SM ={ tree, ring, star, mesh

(grid), complete graphs, bipartite graphs, planar graphs, general graphs}.

Network recognition, as mentioned earlier, is a preprocessing performed in a network before applying a required algorithm. It is clear then that the cost of this preprocessing should not exceed the cost of main algorithm. The basic requirements for network recognition are :

R1. Cost of preprocessing (network recognition algorithm) should be (reasonably) smaller than the cost of the main algorithm.

R2. Preprocessing should be performed by a single algorithm at the termination of which it should be possible to infer whether the network topology represented by G is a member of the set SM.

In other words, R1 specifies when to use the preprocessing and R2 specifies that a unified algorithm should be designed to verify this membership problem. There should not be separate algorithms to recognize each structure in SM.

In this chapter, we develop algorithms to recognize mesh structures. Both centralized and decentralized algorithms are designed and their complexities are established. The problem of network recognition was first introduced by Ramarao [42] where unified algorithms were presented to recognize simple structures that include tree, bipartite graphs, complete graphs, ring, and star. Recognizing mesh and planar graphs were shown to be open problems. We solve an important open problem of recognizing mesh structures. It is established that the solutions for recognizing mesh structures can be unified with the algorithm of Ramarao so that the requirement R2 is satisfied.

6.2 Previous work

The algorithm of Ramarao [42] functions in the following four distinct steps.

Step 1: Algorithm Initiation

Step 2: Processor interaction

Step 3: Local detection

Step 4: Global detection

For centralized algorithms, step 1 is the same. Steps 2 and 3 depend on the structure and the last step is common to all algorithms. The centralized algorithm functions as follows. The algorithm is basically same as the spanning tree construction algorithm discussed in section 3.3. The start processor sends a message to all its neighbors, requesting them to detect a structure. A processor upon receiving such a request verifies whether its local conditions are consistent with that of the structure being sought. If there is some inconsistency, then a reject message is sent to indicate that the network is not of the required structure and the algorithm is terminated. If local conditions are satisfied, then it sends an accept message to its neighbors and upon getting information from all its neighbors, it sends accept message to its parent. If the local conditions at a processor p depend on information of some neighbors, then instead of waiting , p sends out a request to those neighbors, to avoid possibility of deadlocks. When the root of the spanning tree receives all accept messages, it is concluded that the network is of the desired structure that is being recognized.

Local consistency conditions for each topology are listed below.

a. Recognition of tree:

Message: "Color the node red"

Consistent condition: Messages of this type received from exactly one node.

b. Recognition of star:

Message: Depth of a node (distance from central processor)

Consistent condition: Either the depths of all the nodes from which the message is received is 0 or the number of neighbors is 1.

c. Recognition of a Ring:

Message: "Color the node green"

Consistent condition: number of neighbors is 2.

d. Recognition of a complete graph:

Message: depth of a node, number of neighbors

Consistent condition: a. depth of one node from which it has received the message is 0 and b. number of neighbors of the node = number of neighbors of the parent.

e. Recognition of a bipartite graph:

Message: "node color is x" where x is "white" ("red") if the parent color is "red"("white").

Consistent condition: two messages with different colors are not

received.

These techniques are extended to the design of decentralized algorithms. The distributed "divide and conquer" strategy explained in section 3.5 is employed to merge the information from all nodes. The consistency condition is the same as above.

The time and message complexities of centralized algorithm are shown to be O(d(G)) and O(e) respectively and each message is O(log n) bits long. The cost of decentralizing algorithm is higher due to the cost incurred by the divide and conquer algorithm of Gallager et.al. [20]. The time and message complexities of decentralized algorithm are O(nlogn) and O(e+nlogn) respectively where each message is O(logn) bits long.

6.3 Mesh recognition algorithm

We now present distributed algorithms for finding whether a given graph is a mesh. We present both centralized and decentralized algorithms to detect mesh structures and these algorithms can be embedded into the unified algorithms of Ramarao. We develop necessary conditions and the consistency conditions for a graph G to be a mesh.

6.3.1 Necessary and consistency conditions:

For a given nontrivial (n > 4) graph G(V,E), we establish the necessary condition for G to be a mesh. It is obvious that a mesh graph must consist of vertices with degrees of 2,3, and 4 only and that there must be exactly four 2-degree

nodes.

Necessary Condition:

Let a graph G have four 2-degree vertices, m 3-degree, and k 4-degree vertices. We establish conditions for G to be a n1×n2 mesh.

Total number of nodes in G, n	=	n1 n2.
For a mesh, m + k	=	n-4
Number of 3-degree nodes m	=	2(n1 + n2 - 4)
Hence m	=	$2\left(\frac{n}{n2} + n2 - 4\right)$

Solving for n2, we get

$$n2^2 - n2\left(\frac{m}{2} + 4\right) + n = 0 \qquad (6.3.1)$$

The following lemma now follows.

Lemma 6.3.1:

For a nontrivial (n,e) graph G to be a mesh, it is necessary that

a. n2 has an integer solution in equation 6.3.1 and

b. n2|n and

c. deg(v∈ V(G)) ∈ {2,3,4}. □

Consistency condition:

Consider the two adjacent 2-degree nodes v_1 and v_2 in figure 6.3.1. Suppose each node v in G is labeled (m_1,m_2) where m_1 = d(v,v_1) and m_2 = d(v,v_2).

Consider a node v with label (i,j). This node can have at most four neighbors with labels, say, (i1,j1), (i2,j2), (i3,j3) and (i4,j4), listed in some random order. The consistency condition for G to be a mesh, is stated as follows. Proofs and other details are presented later in the chapter.

The labels of the neighbors of v must be:

$i1 = i - 1,\ j1 = j + 1$

$i2 = i - 1,\ j2 = j + 1$

$i3 = i + 1,\ j3 = j - 1$

$i4 = i + 1,\ j4 = j + 1$

And the labels of the four 2-degree nodes must be v_1 = (0,n2); v_2 = (n2,0); v_3 = (n1+n2, n1); and v_4 = (n1, n1+n2). □

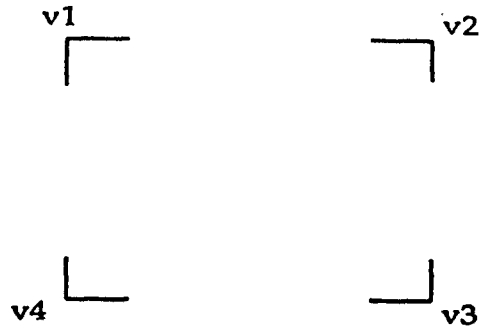

Figure 6.3.1. 2-degree nodes of G.

We now present the algorithm FIND_GRID using the necessary and consistency conditions described above.

6.3.2 Algorithm FIND_GRID:

We now discuss the centralized version of the algorithm FIND_GRID. The algorithm runs in three phases. In the first phase, the necessary conditions are

verified with respect to a given graph G. Second phase locates two vertices v_1 and v_2, some two adjacent 2-degree nodes in G and it also ensures that the shortest distance between adjacent 2-degree nodes along 3-degree vertices satisfies the expected grid dimensions, n1 and n2, established in phase 1. In the third (final) phase, the consistency conditions are verified. It is later shown that a graph G is a mesh if and only if it satisfies the necessary and consistency conditions. The algorithm FIND_GRID is informally described as follows.

PHASE 1: (Verify necessary conditions)

A node s$\in$ V(G) starts the algorithm.

P1.1:construct a spanning tree T of G rooted at the start node.

The spanning tree construction algorithm or the distributed depth first search algorithm, discussed in chapter 3 can be employed. Each message (DISCOVER, RETURN) in the tree construction algorithm has five data fields namely a1, a2, a3, a4, and a5, each of which is logn bits long. Each node before sending a RETURN message updates these fields in the message. Each field represents the following.

a1 = count of total number of nodes in G.

a2 = number of 2-degree nodes.

a3 = number of 3-degree nodes.

a4 = number of 4-degree nodes.

a5 = boolean field, if set to TRUE implies the presence of a node v such that deg(v) $\notin$ {2,3,4}.

The tree construction phase is completed once the root receives messages from all its neighbors. In the process of tree construction, each node earns the degree of its neighbors.

P1.2: Verify necessary conditions.

The root of T verifies the necessary conditions explained in section 6.3.1. If the field a5 in a message is true or if the necessary conditions are not satisfied, then it is inferred that G is not a mesh and the algorithm is terminated. Otherwise the expected dimensions of G are computed. If all the conditions are satisfied, then the root begins phase 2 of the algorithm.

PHASE 2: (Locate 2-degree nodes)

This phase employs two messages namely LOCATE and TRACE. The locate message has two parameters, viz. the grid dimensions. The trace message consists of four parameters, namely f1,f2,n1 and n2; where f1 and f2 refer to node count and flag, respectively, n1 and n2 are the grid dimensions.

P2.1: Locate a 2-degree node, v_1 (say).

This is accomplished by the root sending a LOCATE message to a neighbor in a subtree that contains a 2-degree node. At the termination of phase 1, the root node knows the subtree from where it received a RETURN message, with a nonzero a2 field in the message, indicating the presence of a 2-degree node. The LOCATE message is propagated down the subtree until a 2-degree node is located.

P2.2: Verify boundary and locate v_2.

From v_1, we now trace the boundary of G, by traversing from v_1, moving along 3-degree nodes until we return to v_1.

a. v_1 sends the TRACE message with fields f1 and f2 set as follows. f1 = 1 and f2 = 0. v_1 transmits the TRACE message to a node $v_i \in N(v_1)$ and $\deg(v_i) = 3$.

b. Each 3-degree node v_j that receives the TRACE message propagates the message to a neighbor v_i such that deg(v_i) = 2 or 3 after incrementing the field f1.

c. A 2-degree node v_j, upon receiving the TRACE message does the following computations.

i. If f2 = 0, then set f2 = n1 if f1 = n2; f2 = n2 if f1 = n1; set f1 = 1.

ii. Otherwise, (f2 ≠ 0), if f1 = f2, then

set f2 = n1 if f1 = n2; else set f2 = n2 if f1 = n1. Set f1 = 1.

Clearly, any condition other than those specified above such as f1 ≠ f2, f1 ≠ n1 or f1≠ n2 etc. lead to the conclusion that G is not a mesh. Phase 2 terminates when the TRACE message reaches the originator namely the node v_1.

d. If all the conditions are satisfied, then phase 3 is started by v_1.

PHASE 3: Verify the consistency conditions.

Let v_1 and v_2 be the two 2-degree vertices that are adjacent. All nodes are initially assumed to be in IDLE state.

P3.1: Perform breadth first search with v_1 and v_2 as the roots of the two breadth first search trees.

The algorithm of Awerbuch et.al. [4] can be used to perform distributed breadth first search (BFS). Each BFS establishes a BFS level at each node, L1 and L2 such that L1= BFS level from v_1 and L2 = BFS level from v_2. In other words, for any node v_i with levels L1 and L2, L1 = $d(v_1,v_i)$ and L2 = $d(v_2,v_i)$.

P3.2: A node v_i(L1, L2) with L1 and L2 as defined above, transmits its BFS level information (L1, L2) to all its neighbors.

P3.3: A node v_i transits to SATURATE state if the BFS level information is received from all its neighbors.

P3.4: A node in SATURATE state verifies the consistency condition described in 6.3.2. If the consistency conditions are satisfied at a node v_i, then

a. if v_i is a leaf in T, it sends ACCEPT message to its parent in T.

b. if v_i has received ACCEPT from all its children in T, it sends ACCEPT message to its parent in T. Otherwise, it waits for a message from its children.

If the consistency conditions are not satisfied at any node v_j, it sends REJECT message to all its neighbors in the tree T. A REJECT message amounts to informing the nodes that the graph is not a mesh. The phase 3 terminates

when the root receives either ACCEPT from all its children in T or REJECT message from any of its neighbors.

P3.5: If the root of T receives ACCEPT message from all its neighbors, then it concludes that G is a mesh.

6.3.3 Algorithm analysis and complexity:

In this section, we show the correctness of the algorithm FIND_GRID and establish the message and time complexities of the algorithm. We first show that the there are no deadlocks in the algorithm. By deadlock, we mean a situation in the system which causes nontermination of the algorithm. For example, deadlocks arise due to a node waiting for a message forever, or due to some message getting circulated in a cycle of the graph G without finding a termination for it.

Lemma 6.3.3.1: Algorithm FIND_GRID is deadlock free.

Proof:

From theorem 3.3.1, phase 1 constructs a spanning tree T of G in finite time. In phase 2, the LOCATE message is sent on the tree edges and hence it locates a 2-degree vertex v in $< (n-1)$ time units. The TRACE message can cause deadlock if the message is circulated within a cycle of G repeatedly. However, any 4-degree node that receives this message detects an error situation. Also, any 3-degree node that receives the TRACE message more than once sends a REJECT message to all nodes in G along the tree edges and therefore phase 2

terminates in finite time. The consistency condition verification terminates after each node verifies the test. Since the tree edges are used to convey the ACCEPT/REJECT messages, the root of the tree receives messages from all its children in finite time. □

We now establish that the phase 3 of the algorithm determines correctly whether the given graph is a mesh.

Theorem 6.3.3.2: Suppose the necessary conditions (phase 1 and 2) are satisfied. The graph G is a mesh if and only if the consistency conditions are satisfied.

Proof:

Let v_1 and v_2 be any two adjacent 2-degree vertices of G. Suppose each node v in G is labeled (L1,L2) where L1 = d(v_1, v) and L2 = (v_2, v).

Claim: If the the consistency conditions are satisfied, then label(L1,L2) of each vertex is unique.

Proof:

Proof is based on induction on the distance from a corner vertex, say v_1. Let the (i,j) be the label on a vertex v, where i = d(v_1, v) and j = d(v_2, v). For i = 1, the claim true. For i =1, there must be two neighbors of v_1 , say x and y. If d(x,v_2) = d(y,v_2) = j, then consistency conditions would fail at v_1.

Assume claim true for i = k, k≥ 1.

Consider a node p with label (k+1, j). By consistency conditions, ∃ q∈ V(G), such that label of q = (k,r) where r = j+1 or j-1. Such a neighbor has a

unique label, from hypothesis. We now show that label of p is unique. Suppose there exist two vertices p and p', with labels (k+1,j). If p, $p' \in$ N(q), then consistency conditions would fail at q. Otherwise $\exists\, q' \in p'$ such that the label of q' = (k, r). But, q with label (k,r) is unique. Also, if p' is a three degree node and p is a 4-degree node, then the neighbor set of p' is a subset of that of p and hence p = p'.

We have shown that if the consistency conditions are satisfied, then the labels of each vertex in G are unique. It is easily seen that for a mesh the coordinates are unique and that the distance label can be transformed into the coordinates. Since, the labeling is unique and distance between neighbors differs by 1, G is a mesh.

We now establish the time and message complexities of the algorithm.

Theorem 3.3.3.3:

The algorithm FIND_GRID terminates in $O(n^{1.6})$ time units and requires O(e+nlogn) messages.

Proof: Phase 1 requires O(n) time units and uses at most O(e) messages. Phase 2 locates 2-degree vertices in at most (n-1) time units using < (n-1) messages. The TRACE messages traverses along a path involving 3-degree vertices and it traverses a length of 2(n1+n2) messages in even time. The major contributor to time and message complexities is due to the breadth first search algorithm. Each BFS requires $O(n^{1.6})$ time units, requiring O(e+nlogn) messages which is the

overall complexity of the algorithm. □

6.3.4 Decentralized FIND_GRID algorithm

The design of a decentralized algorithm is easily derived from the centralized algorithm. It is already explained that the algorithm of Gallager et.al. for constructing spanning trees is based on the distributed divide and conquer strategy, and hence supports decentralized construction of a spanning tree of a graph G. Hence, Phase 1 of the centralized algorithm FIND_GRID is modified. A spanning tree of G is constructed using the inductive merge technique [6]. The root at the termination of this algorithm becomes the root of the spanning tree. The rest of the algorithm remains the same thereby retaining the complexity results.

6.4 Discussion

The task of recognizing graph structures is an important activity, but is expensive for certain structures. In fact, from the above algorithm it can be seen that cost of determining mesh structures is more expensive than that for other structures such as trees, rings, etc, that have simple characterizations.

CHAPTER 7

DISTRIBUTED SENSOR NETWORKS

7.1 Introduction

Distributed computing offers several advantages to solve a variety of computational and decision problems. The main advantages include task decomposition, graceful degradation, reduced bandwidth requirements, and the architectures for distributed computing provide a natural way to effect the powerful 'divide-and-conquer' paradigm by solving the subproblems independently.

Based on a distributed processing domain, computations for solving a problem could be performed at various abstraction levels. Each level has a different conceptual content and generates a corresponding set of issues. Each subproblem solution at some particular abstraction level is called a Knowledge Source (KS). Distributed Problem Solving (DPS) is the cooperative solution of problems by a decentralized and loosely coupled collection of knowledge sources, to achieve a global solution to a problem [10, 55, 56]. The knowledge sources cooperate since no one of them has sufficient information and authority to solve the entire problem; mutual sharing of information is necessary to enable the distributed system to produce an answer as a whole. The major difference between distributed processing and distributed problem solving is that in the former, we are mainly concerned with the architecture issues, load balancing, scheduling, processor interconnection, deadlock prevention etc., while in the latter, we are faced

with the problem of finding problem solving methods involving task decomposition, hypothesis , testing and reduction, data fusion etc., on a distributed system domain.

The principle of distributed problem solving is widely used in distributed sensing, especially using the distributed sensor networks. A Distributed Sensor Network (DSN) consists of a set of geographically distributed diverse sensors , a communication network and a set of processing elements; which try to achieve a common goal. Figure 7.1.1 shows the schematic of DSN. The purpose is to obtain an integrated picture of the area covered by the sensors. Motivation for DSN comes mainly from the necessity to increase the sophistication of surveillance systems and tracking mechanisms. Such systems require new architectures and strategies for detecting and tracking multiple targets using data from diverse sensors.

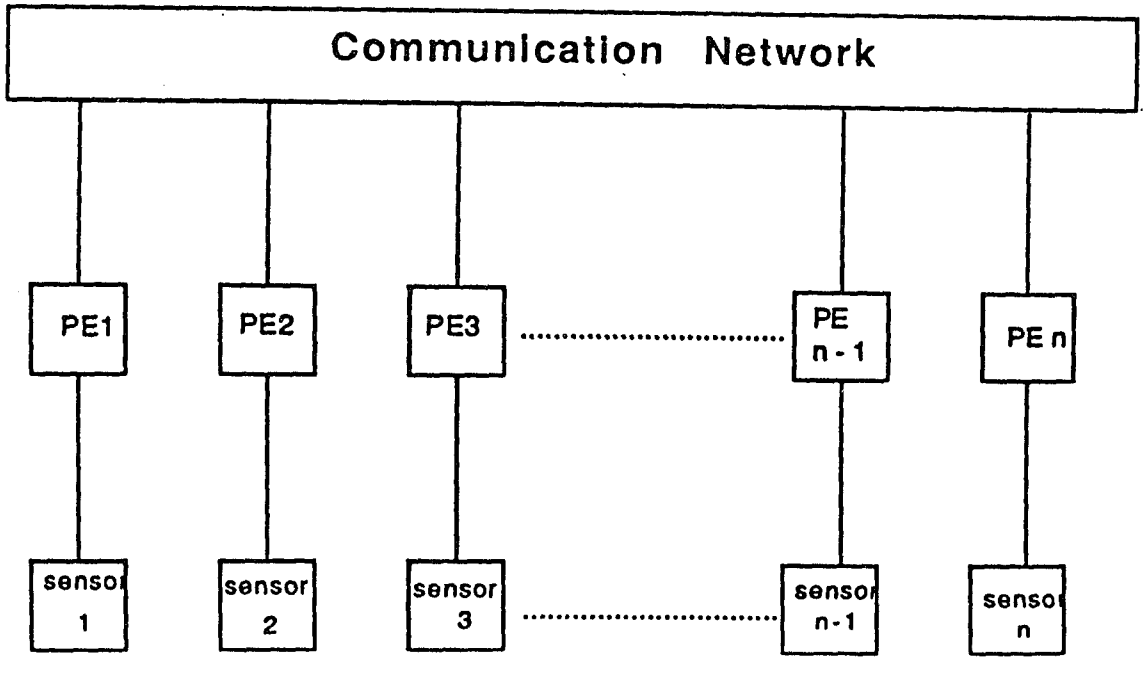

FIGURE 7.1.1. Distributed Sensor Networks- A Schematic.

Design of spatially distributed sensing and decision involves the integration of solutions obtained by solving subproblems in data association, hypothesis and data fusion. It is assumed that the sensors could overlap and information at one sensor is not sufficient to solve the problem. The major reason for distributing sensing, communication, and processing is to achieve a high degree of reliable coverage and survivability of the system.

In this chapter, we discuss the communication and reliability issues in DSN. We define two constraints namely the delay constraint and the reliability constraint, to ensure satisfactory functioning of DSN. Specifically, we present distributed algorithms for determining maximum end-to-end delays in DSN, so that the network meets the delay thresholds. We discuss connectivity requirements for the network to operate in spite of node/link failures.

7.2 Main Issues

One of the key issues in DSN is the communication between processors [62]. The other important issues are the architectures to support real-time distributed inferencing, reliability, signal processing, etc. The principal component of DSN is the coordinated computation amongst the processors which implies that communication amongst processors is the backbone of DSN. Each sensor acts as a knowledge source and communicates to some or all other nodes in the network, to initiate the inference process. The interaction among KS's is quite an expensive operation in distributed systems. It is an important consideration to minimize the interprocess communication and design efficient ways to route the

information in the network.

A processing node receives a bulk of data from the sensor it is associated with, at regular intervals. After some amount of processing at the node, this information is sent to some or all other nodes in the network, depending on the problem solving technique. It is imperative that the information is routed to the destination nodes in an efficient manner since the data generation is repetitive.

Generally, data is transmitted to the destination nodes in packets. Some of the requirements in information routing in DSN are as follows.

i. It is desirable to have the entire information generated by a sensor, in one packet or in fewer packets; otherwise, loss of a packet or delay in receiving it might lead to discarding the entire data. This is a deviation from the packet switching needs in any conventional communication network. The main idea behind such a requirement is to speed up the inference process and to reduce the queue sizes.

ii. In most of the DSN applications, the sensor data will be generated and transmitted in each sensing cycle. Since the data exchange is almost continuous, the communication protocols should be designed such that an explicit ACKNOWLEDGE is not used for each packet. This saves enormous traffic on the network considering the size of DSN.

iii. An immediate effect of not using acknowledge messages is with the old packets that have not yet been processed, but a new packet has arrived. A simple solution to this problem is to ignore the old packets which could be identified by using time-stamps in the message

protocols. However, it is necessary to see that much data is not lost by ignoring old packets and hence it is necessary to route the packets within a maximum allowable time.

iv. DSN is envisaged to operate under hostile environments. It is therefore necessary to employ reliable point-to-point communication protocols. This topic has been well studied in the context of computer networks. These should be adapted to the DSN domains.

7.3 DSN Architectures

A fundamental issue in DSN development is regarding the topological configuration of the network that best fits the application under consideration. This is a complex issue [29, 39] since the interrelated constraints due to task decomposition, node interconnection, communication, fail-safety and integration problems need be considered while designing the network structure. In this section, we briefly review some network architectures for DSN.

Two types of topological configurations for DSN for situation assessment purposes have been studied by Wesson et.al. [61]. First of these is the "Anarchic Committee " organization where each node is able to communicate with the other. This arrangement resembles the "cooperating experts" paradigm in AI. In such an organization, task decomposition and communication issues are resolved dynamically. Each processing node coordinates with some or all other nodes in the network and the solutions are arrived at, after some iterations of information processing, information exchanging and abstractions. No priority is assigned to

any processor and all the nodes operate autonomously.

The other topology studied in [61] is the hierarchical organization where the nodes are organized in strict hierarchies of abstraction levels. Each level will refine the information content of the data received from the lower levels to facilitate global inferencing and transmit the abstracted information to the higher level nodes. Global inference is the responsibility of the nodes in the highest level which can exercise complete control over the network. Task decomposition, authority, and control details are resolved statically to each of the levels. It is shown that the performance of the anarchic committee is superior compared to the hierarchical organization. However, the same conclusion cannot be extended directly to DSN domains due to the small size of their experiment. Also, large bandwidth requirements of the committee structure pose a major problem for large networks such as DSN.

A variation of the hierarchical organization was developed by Iyengar, et.al. [29, 30] , which is a compromise between the hierarchical and the anarchic committee organizations. In this hybrid structure, the sensor domain is partitioned into blocks of sensors called Sensor Cluster Units (SCU), as shown in the figure 7.3.1. Each SCU is organized internally in a hierarchical manner as shown in figure 7.3.2. SCU's are formed in such a way that integration of information from each sensor in an SCU is performed in the respective hierarchical system. Each 'commander' node (the node at the highest level in SCU) of an SCU is connected to all other peer nodes in the sensor domain. In other words, an anarchic committee of SCU commander nodes is formed. Such an organization could be

effectively used for situation assessment in a known problem domain. The hybrid network organization thus incorporates the advantages of both hierarchical and committee organizations and provides a natural way to implement the divide-and-conquer strategy.

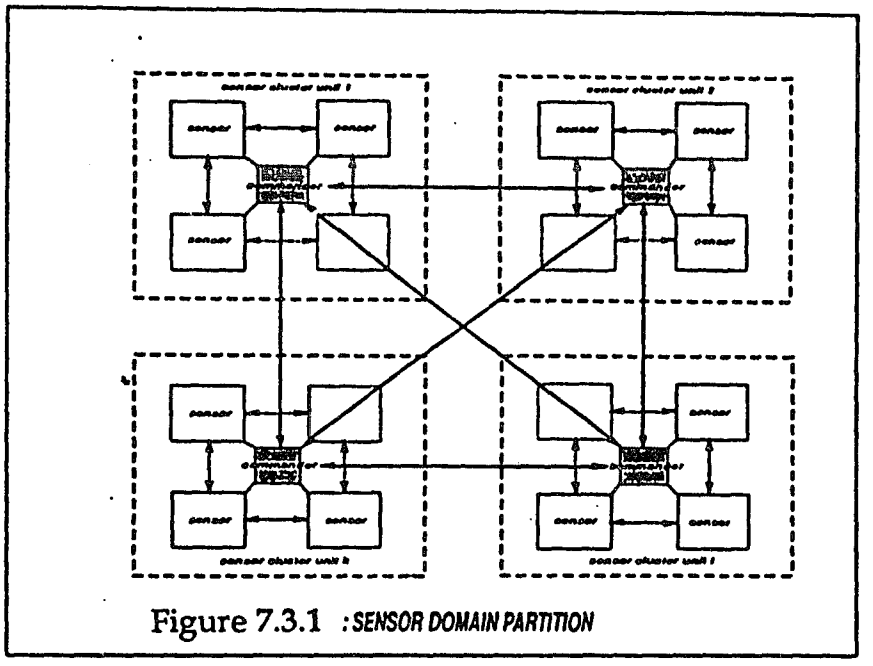

Figure 7.3.1 : SENSOR DOMAIN PARTITION

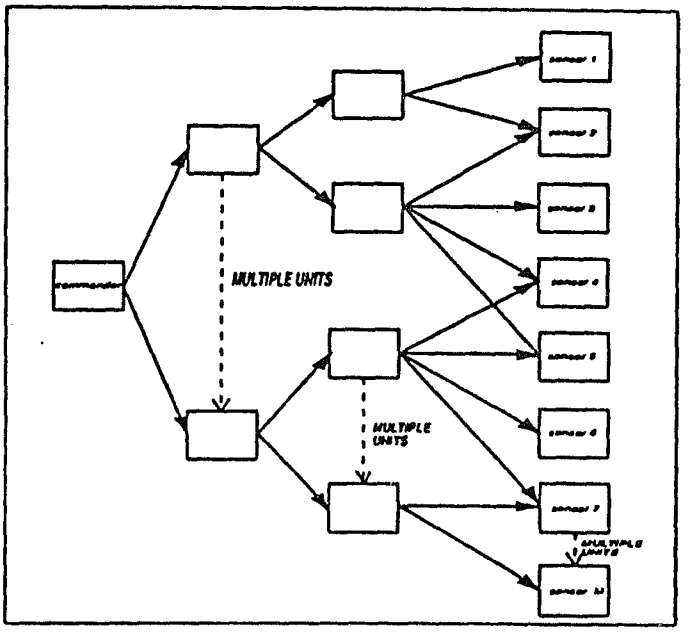

Figure 7.3.2 : SENSOR CLUSTER UNIT ORGANIZATION

7.4 Communication and Reliability

Performance of any coordinated computing system depends on the communication and computational speeds. Communication poses a major problem since the time needed for communication is large in large networks. In DSN, the sensors generate data repetitively at regular intervals. It is therefore essential to ensure that the data is delivered to the destination node in finite time, before the data of the next cycle arrives at the node. Hence there is an upper bound on the maximum allowable transmission time in the network. Also, considering the environment that DSN operates in, it is necessary to examine fail-safety requirements. In this section, we investigate mainly two constraints for communication, viz. the delay constraint and the reliability constraints. We present distributed algorithms for determining the maximum transmission delay in the network and study the connectivity requirements to achieve reliable communication in DSN. We also study the impact of node/link failures on the delay.

7.4.1 Delay Constraints.

When a node in DSN needs to transmit information to other nodes in the network, the communication should be completed before another (set of) data of the next cycle arrives at the node. The problem could be stated slightly differently. A node n_i needs to transmit to other nodes, a unit of information within time t_d where t_d is the maximum permissible transmission delay. The question is whether this is possible in the present configuration of the network. From time to time this check need be done in the network for proper functioning of DSN.

The factors that affect transmission delay in DSN include the capacity of the channel, number of lines on the route, propagation delay, and the packet size. From queuing theory results [8] , we have an approximation for the mean total delay T,

$$T = L/\{ C(1-\rho) \} + t_p$$

where

L= mean length of frames

C= channel capacity

ρ = load on the line

t_p = propagation delay.

For a network, the parameters L,C and ρ are generally known to an acceptable degree of approximation. For a dynamic network, it is necessary to determine t_p. The propagation delay is dependent mainly on the length of the route and hence the maximum end-to-end distance is the worst case measure of the propagation delay.

A network could be represented as a graph G(V(G), E(G)) where V(G) is the set of vertices in the graph denoting the nodes in the network and E(G) is the set of edges in the graph denoting the communication links of the network. We assume that the links are bidirectional and the network does not experience any failure during the execution of the algorithm. It is assumed that the messages are delivered to the other nodes in finite time and there is no loss of messages. This is identical to the models of distributed systems studied earlier.

Diameter of the graph underlying the network gives a measure of the maximum end-to-end transmission delay. We now present distributed algorithms to determine the diameter of the graph underlying a network.

First we consider asynchronous tree network and present diameter finding algorithms considering two cases; when one node starts the algorithm, and when multiple nodes start the algorithm. We discuss the extension of these algorithms to general networks.

7.4.2 Distributed diameter finding algorithms:

We now present two algorithms, SDIA and MDIA which determine the diameter of asynchronous tree networks when an arbitrary node starts the algorithm and when an arbitrary number of nodes start the algorithm, respectively. The main idea behind these algorithms is to determine two largest heights h_1 and h_2, say, of each node and the node with the largest $h_1 + h_2$, determines the diameter of the tree. If the start node lies on the diameter path, then it computes the diameter, otherwise it receives the diameter information from a node that computed it. These algorithms are based on the SDCM and MDCM algorithms discussed in chapter 5. We assume that a node v is in TERMINAL state, after the execution of a basic algorithm SDCM or MDCM.

A. Algorithm (SDIA) :

The algorithm SDIA determines the diameter of the tree when a single node starts the algorithm and is based on the algorithm SDCM. The computations per-

formed at each node and the actions at each node in TERMINAL state are described below.

Each leaf node sends the RETURN(p_1, p_2) message with parameters $p_1 = p_2 = 0$. Each node upon receiving the RETURN (p_1, p_2) message performs the following computations.

i. increments p_1 by one.

ii. computes the two largest p_1's , h_1, and h_2 (say) computed so far; (both h_1 and h_2 are initially 0)

iii. computes pd = max(pd, p_2, $h_1 + h_2$) which is the maximum end-to-end distance at that node, known so far. (pd is initially set to 0)

When an internal node sends the RETURN(p_1, p_2) message to a neighbor, then the parameters are set as: $p_1 = \max(h_1, h_2)$ and $p_2 = \max$ (pd, $h_1 + h_2$), thus computed at that node.

If the start node is in TERMINAL state, then the algorithm is terminated with the start node computing the diameter = max (pd, $h_1 + h_2$). Otherwise, the node i in TERMINAL state sends SETDIA(max(pd,h_1+h_2)) to its neighbor that caused the state transition in node i (from IDLE to WAIT, or IDLE to TERMINAL.) (note that this situation occurs when a leaf starts the algorithm in tree structures similar to those in figures 1 and 2.) The process is continued until the start node receives the SETDIA message upon which it records the parameter pd in SETDIA as the diameter of the tree, and exits the algorithm.

We now present the complexity analysis and proof of correctness for the

algorithm SDIA. Suppose a node $s \in V(T)$ starts the algorithm SDIA.

Lemma 7.4.2.1:

The algorithm SDIA correctly determines the diameter of the tree and terminates in finite time.

Proof:

The EXPLORE message initiated at some start node s, traverses down all the subtrees rooted at s, to each leaf. Each internal node i computes the maximum end-to-end distance of the subtree rooted at i in T(s) (i.e. pd of the return message). Since the diameter of the graph is the maximum end-to-end distance, some internal node computes it. If s is on the diameter path, then s computes it, otherwise, s receives the computed diameter information from a node on the diameter path through its neighbor.

Since the RETURN message moves up the tree starting from the leaf, the start node receives the return message from all its neighbors, thus terminating the algorithm in finite time. □

Let h(T) denote the height of the tree T and let h_{max} = max { h (T(j,s))| for all T(j,s) ∈ T(s)}.

Theorem 7.4.2.1:

Suppose a node s starts the algorithm SDIA. Then, s determines the diameter of the tree T in at most $2h_{max}$ units of time, if each message is delivered over a link in at most one unit of time.

Proof:

The message EXPLORE moves down the tree rooted at s and the RETURN message retraces the path of EXPLORE message. Since each internal node broadcasts the EXPLORE message, the maximum time it takes for the EXPLORE message to reach a leaf is to the one at the farthest distance from s. Hence, the total time taken for finding diameter is 2h(T(i,s)) where T(i,s) is the subtree rooted at s which has the maximum height. □

Theorem 7.4.2.2:

The algorithm SDIA finds the diameter of the tree T using exactly 2(n-1) messages and is optimal in message complexity.

Proof: It is clear from the algorithm SDIA that the EXPLORE message travels on all the edges of the tree to the leaf and the RETURN message retraces the EXPLORE message. Thus, there are two messages on each edge and hence the algorithm uses exactly 2(n-1) messages.

Any diameter finding algorithm, the message has to be sent over each link of the tree and hence the problem of finding diameter has a message complexity lower bound of O(n). The algorithm is hence message optimal. □

B. Algorithm MDIA:

We describe the diameter finding algorithm MDIA based on the basic algorithm MDCM, when multiple nodes start the algorithm.

The RETURN message consists of two parameters p_1 and p_2. Each leaf sends RETURN message with $p_1 = p_2 = 0$; Each node i upon receiving the RETURN(p_1, p_2) message, performs the following computations.

i. increments p_1.

ii. computes the two largest p_1's , h_1 and h_2 (say), received. (both h_1 and h_2 are initially 0 .)

iii. computes pd = max(pd,p_2, $h_1 + h_2$) which is the maximum end-to-end distance at the node. (pd is initially set to zero.)

When an internal node sends the RETURN(p_1, p_2) message, the parameters are set as: $p_1 = \max(h_1, h_2)$ and $p_2 = \max(\text{pd}, h_1 + h_2)$, thus computed at that node.

As stated earlier, a node reaches the TERMINAL state if it has received RETURN message from all its neighbors. Hence, the TERMINAL node has either computed or received the maximum end-to-end distance information which is the diameter of the tree. A node in TERMINAL state broadcasts SETDIA(pd) to all its neighbors and exits the algorithm.

1. When a node i in ACTIVE state receives the SETDIA(pd) message from a neighbor j,

1.1 broadcasts the received message to all nodes $k \in N(i)-\{j\}$, and

1.2 if $i \in S$, then it records the diameter of the tree as pd.

1.3 the node i then exits the algorithm.

2. When a node i in TERMINAL state receives the SETDIA (pd) message

from a neighbor j, the message is ignored.

{ this can happen when two adjacent TERMINAL nodes exist. }

Clearly, the algorithm MDIA has the terminating property whereby each node will know when the algorithm has terminated. Suppose a set of node S ={ $s_1, s_2, .. , s_k$ }, S $\subset$ V(T), start the algorithm.

Lemma 7.4.2.4: The algorithm MDIA correctly finds the diameter of the tree network and terminates, in finite time. At the termination of the algorithm, each start node has learned the diameter information.

Proof:

The EXPLORE message is initiated by all the start nodes S =$\{s_1, s_2, .., s_l\}$, S $\subseteq$ V(T). Each node $s_i \in$ S functions as if it is the only start node. A node sends the RETURN message only when it has received RETURN from all its neighbors except one. Hence, when a node receives the RETURN message from its neighbor, it is implied that all the necessary computations are completed in the subtree rooted at the neighbor. Each internal node determines the maximum end-to-end distance in the subtree. Some node n_j will eventually receive the RETURN message from all its neighbors and hence computes the diameter of the tree. All the nodes terminate after receiving the SETDIA message and hence the algorithm terminates in finite time. □

Let h(T) denote the height of the tree T and d the diameter of the tree. Let h_{max}(S) = max{h(T_i): for all i $\in$ S}

Theorem 7.4.2.4:

Suppose nodes $S = \{s_1, s_2, .., s_l\}$, $1 \leq l \leq n$, and $S \subseteq V(T)$ start the algorithm MDIA. Then, each of them will learn the diameter of the tree in at most $2h_{max} + D(T)$ units of time if each message is delivered in one unit of time.

Proof: In the worst case, the EXPLORE message travels from one end of the tree to the other, even if multiple nodes started the algorithm. The maximum distance traversed by the EXPLORE message originating at a start node s_i is equal to the maximum height of the subtree rooted at s_i. The RETURN message retraces this path and these two messages together account for the first term in the time complexity. Once a node reaches the TERMINAL state, it broadcasts the SETDIA message to all other nodes in the network which traverses less than the diameter path length which accounts for the second term in the time complexity relation.

Theorem 7.4.2.5: The algorithm MDIA finds the diameter of the tree using at most 4(n-1) messages.

Proof: In the worst case, all nodes in the tree might start the algorithm simultaneously and each will send the EXPLORE message to their neighbors, except the leaf nodes which send the RETURN message. This amounts to 2(n -1) messages. There will be (n-1) RETURN messages before the node can get to the TERMINAL state. A node in TERMINAL state broadcasts the SETDIA message which amounts for (n-1) messages and hence the message complexity of the algorithm MDIA is at most 4(n-1). □

Finding diameter distributively of an arbitrary graph is expensive in terms of communication complexity. The algorithm to determine the center of a graph in [13] could be employed to determine the diameter of an arbitrary graph. The algorithm is outlined as follows.

i. Construct a spanning tree of the graph rooted at a node s, using any of the existing algorithms explained in chapter 3.

ii. Each node in the tree then initiates a shortest path finding algorithm and determines its eccentricity.

iii. Each node then computes its maximum end-to-end distance and sends this information to the root.

iv. The root node, upon receiving the end-to-end distance information from each of the nodes in the network, computes the largest of these which is the diameter of the graph. Then the diameter information is broadcast to all other nodes in the network.

The communication complexity of the algorithm is computed as follows. The first step requires O(e) messages and the shortest path finding algorithm initiated at each node requires O(e) messages. Thus, total number of messages required to compute the shortest paths by all nodes is O(n.e). It needs $O(n^2)$ messages to perform step iii and the last step of the algorithm requires O(n) messages. Hence, the message complexity is O(n.e) which in the worst case is $O(n^3)$.

7.4.3 Reliability Constraints:

A vital consideration for automatic routing and proper functioning of DSN is the survivability of the network. Nodes and links in DSN may fail due to several reasons considering the fact that DSN works in hostile environments. It is therefore necessary to maintain the message flow in the network. In this section, we describe the connectivity requirements that need be taken into account while designing the topological structure for DSN that will achieve fail-safety. We also discuss the importance of these failures on the end-to-end delay in the network.

When a node/link fails, it is necessary to find alternate routes to route messages. Distributed protocols for such dynamic routing are discussed in the next section. However, it is necessary that the topological configuration of the network has sufficient connectivity such that when some links and/or nodes fail, the network remains connected. This requirement should be taken into consideration while designing network topologies. Suppose at most L links fail in the network. Then the graph underlying the network must be (L+1) edge-connected for the network to remain connected despite the failure of L links. Similarly, for the network to remain connected if a k nodes fail, then the graph must be (k+1)-connected. Thus, it is possible to keep the network connected despite link/node failures by increasing the connectivity of the graph. However, a node/link failure can increase the diameter of the graph, thus increasing the end-to-end delay in the network.

Suppose D(G) is the diameter of the graph G underlying a network. We assume that we are given a k-connected graph of order n, and that there will be at most m node failures at any instant of time in the network. Let us now derive a condition for k such that under m node failures, the diameter of the network does not exceed the maximum permissible diameter D', for which the transmission delay does not exceed t_p. In [8] , it is shown that deletion of m nodes in a k-connected graph of order n (m<k), the diameter of the resulting graph D' is bounded by;

$$D' \leq \lfloor \frac{n-m-2}{k-m} \rfloor + 1.$$

It is clear that for D' to be within the delay constraint, under a fixed m, the connectivity k of the graph could now be computed using the above inequality, such that not only network connectedness is maintained, but also the node failures do not result in unacceptable delays. Also, for a λ edge-connected graph of order n, deletion of λ-1 results in a graph of diameter D' such that,

$$D' \leq \lambda D \div \lambda - 1,$$

where D is the diameter of the original graph.

We have shown the effect of link/node failures on the connectedness of the graph and the delay in the network. These results provide means for retaining the network connected and also to ensure that the delay constraints for DSN are met with.

CHAPTER 8

CONCLUSIONS

A distributed system as envisaged by Enslow [18] assumes complete independence in data, control, and processors. Distributed algorithms for investigating the topological parameters are extremely important for solving a variety of problems in distributed systems. We have presented efficient distributed algorithms for two generic problems related to finding topological parameters in distributed computing, namely the network traversal problems and the distributed graph problems.

We have presented an efficient algorithm for distributed depth-first traversal of asynchronous communication networks. Performance of our algorithm is shown to be superior to the existing algorithms. More importantly, it is observed that our algorithm is more suitable for weighted networks, since the message protocols can be restricted to a path of smallest weight. Importance of network learning algorithms is well known in the area of distributed computing. We have presented two learning algorithms based on our distributed depth first search algorithm which improve the message complexity of such algorithms by O(n). These two algorithms solve the problem of finding connectivity and global topology, in asynchronous communication networks.

Interesting results have been presented for network location problems. Besides showing a counterexample to the only existing work on this topic (

Korach et.al. [34]), we have presented a novel algorithm to determine the centers and medians of asynchronous tree networks. This is the only decentralized algorithm for finding centers and medians in the literature. Also, extensions of this algorithm to weighted trees brings out an interesting result that the location of medians in a tree is independent of whether the tree is weighted or unweighted.

Network recognition is an important activity in acquiring topological knowledge of a distributed system. Towards this end, we have presented a novel solution to an open problem, i.e., the problem of recognizing mesh (grid) structures, in asynchronous networks.

Future research directions:

1. In the model for distributed system, we assumed that the network is reliable and that there is no node/link failures. However, such an assumption is not valid for a real system where node/link failures are expected. New techniques are needed to transform the results for static networks to dynamic networks.

2. For synchronous networks, it is shown that the lower bound for message complexity for distributed depth first search is 2e. It will be interesting to investigate if this bound could also be met for an asynchronous network.

3. The network location algorithms presented in this dissertation are efficient for tree networks. It is also shown that the algorithm for finding centers and

medians for general networks is expensive in terms of communication complexity. An open research problem is to find approximate solutions to these problems.

4. In the network location algorithm, we have considered the problem of finding one center of a network. The problem of finding p-centers and p-medians distributively is an open problem. Our algorithms can be extended to determine the p-centers and p-medians for tree networks. However, the problem of finding p-centers and p-medians for asynchronous networks is NP-complete.

5. A planar graph is an important structure for which fast algorithms are available in the literature for several problems. It is an important problem to recognize planar graphs distributively.

References

[1] AHO, A.V., HOPCROFT, J.E., AND ULLMAN, J.D., "The design and analysis of computer algorithms," *Addison-Wesley Publishing Company* , 1974.

[2] ANGLUIN, D., "Local and global properties in networks of processors," *Proc. ACM STOC, New York*, pp. 82-93, 1980.

[3] AWERBUCH, B., "A new distributed depth-first-search algorithm ," *Inform. Process. Lett.*, vol. 20 (3) , pp. 147-150, 1985 .

[4] AWERBUCH, B. AND GALLAGER, R.G., "A new distributed algorithm to find breadth first search trees," *IEEE Trans. Information Theory*, vol. IT-33 (3), pp. 315-322, May 1987.

[5] AWERBUCH, B., GOLDREICH, O., PELEG, D., AND VAINISH, R., "A Tradeoff Between Information and Communication in Broadcast Protocols," *Proc. Workshop on VLSI architecture and algorithms, Greece, Springer-Verlag LNCS in Computer Science*, pp. 369-379, June-July 1988.

[6] AWERBUCH, B., HANDLER, G.Y., AND MIRCHANDANI, P., "Location on Networks: theory and applications," *MIT Press, Cambridge, Mass.*, pp. 230-240 , 1979.

[7] BOLLOBAS, B., "Extremal graph theory," *Academic Press, New York*, 1978.

[8] BOND, J. AND PYERAT, C., "Diameter vulnerability in networks," *in Graph theory with application to algorithms and computer science*, vol. Ed.Y.Alavi et.al., John Wiley and Sons, pp. 125-149, 1985.

[9] BUCKLEY, F. AND HARARY, F., "Distance in graphs," *Addison-Wesley Publishing Company*, 1990.

[10] CHANDRASEKARAN, B., "Natural and Social system metaphors for Distributed Problem Solving: Introduction to the issue," *IEEE trans. Syst. Man. Cyber.*, pp. 1-4, Jan 1981.

[11] CHANDY, K.M. AND LAMPORT, L., "Distributed snapshots: Determining global states of distributed systems," *ACM Trans. on Computer Systems*, vol. 3 (1), pp. 63-75, Jan 1985.

[12] CHANG, E.J.H., "Echo algorithms: depth parallel operations on general graphs," *IEEE Trans. on Software Engineering*, vol. SE-8 (4), pp. 391-401, 1982.

[13] CHEUNG, T., "Graph traversal techniques and the maximum flow problem in distributed computation ," *IEEE Trans. Software Engineering*, vol. SE-9 (4), pp. 504-512, 1983 .

[14] CIDON, I., "Yet another distributed depth-first-search algorithm," *Inform. Process. Lett* , vol. 26 (6), pp. 301-305, Jan 1988 .

[15] DALAL, Y.K., "A distributed algorithm for constructing minimal spanning trees," *IEEE Trans. Software Engineering*, vol. SE-13 (3),

pp. 398-405, March 1987.

[16] DECHTER, R. AND KLEINROCK, L., "Broadcast Communications and Distributed Algorithms," *IEEE Trans. Computers*, vol. c-35, 3, pp. 210-218, March 1986.

[17] DIJKSTRA, E.W. AND SCHOLTEN, K., "Termination by diffused computations," *Inform. process. lett.*, 1980.

[18] ENSLOW, P.H., "What is a distributed system?," *Computer*, pp. 13-21, January 1978.

[19] FREDERICKSON, G., "A single source shortest path algorithm for a planar distributed network," *Proc. Symp. Theoret. Aspects Comput. Sci.*, January 1985.

[20] GALLAGER, R.G., HUMBLET, P.A., AND SPIRA, P.M., "A distributed algorithm for minimum-weight spanning trees," *ACM trans. Programming languages and systems*, vol. 5 (1), pp. 66-77, Jan 1983.

[21] GARCIA-MOLINA, H., "Broadcast algorithms," *IEEE trans. Computers*, 1982.

[22] GAREY, M.R. AND JOHNSON, D.S., "Computers and Intractability, A guide to the Theory of NP-Completeness," *W.H. Freeman and Company, New York*, 1979.

[23] HARARY, F., "Graph Theory," *Addison-Wesley, Reading, Mass.*, 1969.

[24] HARARY, F. AND NORMAN, R.Z., "The dissimilarity characteristic of Husimi trees," *Ann. Math.*, vol. 58, pp. 134-141, 1953.

[25] HEART, F.E., ORENSTEIN, S.M., CROWTHER, W.R., AND WALDEN, D.C., "The interface message processor for the ARPA computer network," *in Proc. Spring AFIPS conf.*, 1970.

[26] HOROWITZ, E. AND SAHNI, S., *Fundamentals of Computer Algorithms*, Computer Science Press Inc , 1984.

[27] HUANG, S.T., "A new distributed algorithm for the biconnectivity problem," *Proc. Intl. Conf. on Parallel Processing*, vol. 3, pp. 106-113, 1989.

[28] HUMBLET, P.A. AND SOLOWAY, S.R., "Topology Broadcast Algorithms," *Computer Systems and ISDN Systems*, vol. 16, pp. 179-186, 1988-1989.

[29] IYENGAR, S.S. AND SHARMA, M.B., "Information Routing in Distributed Sensor Networks," *in Proc. Indo-US Workshop on High Speed Digital Processing, New Delhi*, Nov. 27-29 1989.

[30] IYENGAR, S.S., SHARMA, M.B., AND KASHYAP, R.L., "Communication and reliability issues in Distributed Sensor Networks," *submitted to IEEE Trans. Software Engg.*, October 1989.

[31] JORDAN, C., "Sur les assemblages de lines," *J.Reine Agnew. Math.*, vol. 70, pp. 185-190, 1869.

[32] KARIV, O. AND HAKIMI, S.L., "An algorithmic approach to network

location problems: I. The p-centers," *J. SIAM Appl. Math.*, vol. 37 (3), pp. 513-538, 1979.

[33] KARIV, O. AND HAKIMI, S.L., "An algorithmic approach to network location problems: II. The p-medians," *J. SIAM Appl. Math.*, vol. 37 (3), pp. 539-560, 1979.

[34] KORACH, E., ROTEM, D., AND SANTORO, N., "Distributed algorithms for finding centers and medians in networks," *ACM Trans. Prog. Lang. Syst.*, vol. 6 (3), pp. 380-401, July 1984.

[35] KUMAR, D., IYENGAR, S.S., AND SHARMA, M.B., "Corrections to a distributed depth first search algorithm," *to appear in Inform. Process. Lett.*, 1990.

[36] LAKSHMANAN, K.B., MEENAKSHI, N., AND THULASIRAMAN, K., "A time optimal message-efficient distributed algorithm for depth-first-search," *Inform. Process. Lett.*, vol. 25 (2), pp. 103-109, May 1987.

[37] LAMPORT, L., "Time, clocks, and ordering of events in a distributed system," *Communications of the ACM*, vol. 21, no. 7, pp. 63-75, July 1978.

[38] LEEUWEN, J. VAN, "Distributed algorithms, 2nd Intl. Workshop ," *Lecture Notes in Computer Science (312), Springer-Verlag*, vol. Amsterdam, The Netherlands, July 1987.

[39] LESSER, V., CORKILL, D., PAVLIS, J., AND BROOKS, R., "A high

level simulation test bed for cooperative distributed problem solving,'' *Proc. DSN workshop, MIT Lincoln Lab.*, pp. 247-270, Jan 1982.

[40] LIESTMAN, A.L., ''Fault-tolerant broadcast graphs,'' *NETWORKS*, vol. 15, pp. 159-171, 1985.

[41] LYNCH, N. AND FISCHER, M., ''On describing the behavior and implementation of distributed systems,'' *Theoret. comp.sci.*, vol. 13, pp. 17-43, 1981.

[42] RAMARAO, K.V.S., ''Distributed algorithms for network recognition problems,'' *IEEE Trans. Computers*, vol. TC-38 (9), pp. 1240-1248, September 1989.

[43] RAMARAO, K.V.S., DALEY, R., AND MELHEM, R., ''Message complexity of the set intersection problem,'' *Inform. Process. Lett.*, vol. 27 (4), pp. 169-174, April 1988.

[44] RAYNAL, M., ''Introduction to Distributed algorithms ,'' *Distributed algorithms and protocols, John Wiley and Sons*, pp. 1-15, 1988.

[45] RAYNAL, M., ''Networks and distributed computation: concepts, tools and techniques,'' *The MIT Press, Cambridge*, 1988.

[46] REIF, J.H., ''Depth-first-search is inherently sequential,'' *Inform. Process. Lett.*, vol. 20 (5), pp. 229-234, June 1985.

[47] SCHNIEDER, B. AND , L. LAMPORT, ''Paradigms for distributed programs,'' *Lecture notes in computer science - Distributed Systems,*

Springer-Verlag, pp. 431-468, 1985.

[48] SEGALL, A., "Distributed Network Protocols," *IEEE Trans. Inform. Theory*, vol. IT-29, 1, pp. 23-35, January 1983.

[49] SHARMA, M.B., CHEN, J., AND IYENGAR, S.S., "Efficient distributed algorithms for finding topological parameters in communication networks," *submitted to ACM Trans. Prog. Lang. Syst.*, Nov 1989.

[50] SHARMA, M.B., IYENGAR, S.S., AND MANDYAM, N.K., "An efficient distributed depth-first-search algorithm," *Inform. Process. Lett.*, vol. 32 (4), pp. 183-186, September 1989.

[51] SHARMA, M.B., MANDYAM, N.K., AND IYENGAR, S.S., "An optimal distributed depth-first-search algorithm," *Proc. ACM National Conference on Computer Science*, vol. Louisville, KY, Oct 19-21, 1988.

[52] SLATER, P.J., "Medians of arbitrary graphs," *J.Graph Theory*, vol. 4, pp. 389-392, 1980.

[53] SLATER, P.J., "Accretion centers: a generalization of branch weight centroids," *Discrete Appl. Math.*, vol. 3, pp. 187-192, 1981.

[54] SLATER, P.J., "Locating central paths in a graph," *Transportation Sci.*, vol. 16, pp. 1-18, 1982.

[55] SMITH, R.G., "A framework for distributed problem solving," *UMI press, Michigan*, 1981.

[56] SMITH, R.G., "Frameworks for cooperation in distributed problem solving," *IEEE trans. Syst. Man. Cyber.*, pp. 61-70, Jan 1981.

[57] SPINELLI, J.M. AND , R.G. GALLAGER, *IEEE Trans. Communications*, vol. 37, 5, pp. 468-474, May 1989.

[58] TARJAN, R.E., "Depth first search and linear graph algorithms," *SIAM J. Comput.*, vol. 1 (2), pp. 146-160, 1972.

[59] TOPKIS, D.M., "All-to-all Broadcast by Flooding in Communication Networks," *IEEE Trans. Computers*, vol. 38, 9, pp. 1330-1333, September 1989.

[60] TRUSZCZYNSKI, M., "Centers and centroids of unicyclic graphs," *Math. Slovaca.*, vol. 35, pp. 223-228, 1985.

[61] WESSON, R., "Network structures for Distributed Situation Assessment," *IEEE trans. Syst. Man. cyber.*, pp. 5-23, Jan 1981.

[62] YEMINI, Y., "DSN, an attempt to define issues," *Proc. DSN Workshop, CMU*, pp. 53-66, 1978.

[63] ZAKS, S., "Optimal distributed algorithms for sorting and ranking," *IEEE Trans. Computers*, vol. C-34 (4), pp. 376-379, April 1985.

[64] ZELINKA, B., "Medians and peripherians of trees," *Arch. Math. (Brno)*, vol. 4, pp. 87-95, 1968.

[65] ZHU, Y. AND CHEUNG, T., "A new distributed breadth-first-search algorithm," *Inform. Process. Lett.*, vol. 25 (5), pp. 329-333, July 1987.

APPENDIX

A3.4 Formal description of the algorithm DDFS.

Variables for node i:

$father_i$ = Father of node i in DFS.

son_i = set of children of node i in DFS.

$maxadj_i$ = largest $j \in adj_i$.

source = Marks the root node. set to -1 initially.

adj_i = Adjacency list of node i.

$leafnode_i$ = set to FALSE, indicates whether i is a leaf node in DFS or not.

Algorithm for node i ≠ root:

<1>. upon RCV (MSG_j^i)

<2>. if MSG.HEAD = DISCOVER, then

<3>. if $j \in son_i$, SEARCH;

else

<4>. $father_i$ = j; MSG.NLIST[i] = VISITED;

if $maxadj_i$ > |MSG.NLIST|, then

extend MSG.NLISt to size of $maxadj_i$;

initialize added bits to UNVISITED;

SEARCH;

end;

Algorithm for node i = root node :

<5>. upon RCV (START msg) from external world,

<6>. source = i; create MSG with bit-array size k = max($maxadj_i$, i);

MSG.HEAD = DISCOVER; MSG.NLIST[j] = UNVISITED for $0 < j \leq k$;

MSG.NLIST[i] = VISITED ;

SEARCH;

Procedure SEARCH;

<7>. if $\exists$ k, s.t. MSG.NLIST[adj_i[k]] = UNVISITED then

<8>. then $son_i = son_i \cup adj_i$ [k]; send MSG to node k.

<9> else

<10>. if source $\neq$ i then send MSG to $father_i$

if $son_i \neq \emptyset$ then $leafnode_i$ = TRUE ;

<11>. else

<12>. STOP; termination of algorithm;

A4.3 Formal decsription of Algorithm MCT:

We add a new message called CONNECT with the same message structure as the DISCOVER message of DDFS. The root node sends the CONNECT message when algorithm DDFS terminates, to inform the other nodes in the tree of the nodenames that are connected. We modify line <12> and add line <4a> following line <4> in the DDFS algorithm. The additional local variables at each node i are listed below.

For node i,

$cn[]$ = { j ∈ V| path (i,j) exists }, Initialized to nil;

---

i. For each node i ≠ root, add after line <4>,

<4a>. if MSG.HEAD = CONNECT then

set $cn[k]$ = CONNECTED whenever MSG.NLIST[k] = VISITED

for all k ≤ |MSG.NLIST|;

if i ≠ *leafnode*

send MSG to k,for all k ∈ $\{son_i\}$

ii. For node i = root, replace line number <12> in DDFS by :

<12>. MSG.HEAD = CONNECT;

<12a>. set $cn[k]$ = CONNECTED whenever MSG.NLIST[k] = VISITED

for all k ≤ |MSG.NLIST|;

<12a>. send MSG to all k $\in$ son_i.

end.

A4.3 Formal description of algorithm MLT:

We modify the algorithm DDFS and employ additional messages namely ADJACENT and RADJACENT. Structure of these two messages is similar to that of the DISCOVER message of DDFS. The ADJACENT message contains the message header, nodename and its adjacency list. The adjacency list of a node i is an n-bit array with bit j set to TRUE if $j \in adj_i$. The RADJACENT message is similar to the ADJACENT message and contains an additional field for sending total number of nodes connected in the network and this message is initiated only by the root node.

Additional variables at node i

GR [] = adj_j for all $j \in V$; Initialized to nil.

TN = Total connected nodes in network.; set to -1;

RN = local counter; set to 0;

SF = flag, set to 0; indicates whether node i sent adjacent message or not.

Replace line <12> in DDFS by:

<12>. for node i = source (root node)

TN = number of VISITED bits in MSG.NLIST[];

create MSG;

MSG.HEAD = RADJACENT; MSG.ID = i; MSG.NUM = TN;

set MSG.ALIST[j] = TRUE for all j ∈ adj_i;

send MSG to all j ∈ son_i;

SF = TRUE;

For any node i (including root):

<13>. upon RCV(MSG_j^j)

if MSG.HEAD = ADJACENT or RADJACENT then

$RN = RN + 1$;

if i ≠ *leafnode* then

send MSG to all k ≠ j, k ∈ { $son_i \cup father_i$ }

GR[MSG.ID] = MSG.ALIST[];

if MSG.HEAD = RADJACENT then TN = MSG.NUM;

if (! SF) then

create MSG;

MSG.HEAD = ADJACENT; MSG.ID = i;

set MSG.ALIST[k] = TRUE for all k ∈ adj_i;

send MSG to all k ∈ { $son_i \cup father_i$ };

SF = TRUE;

if $RN = TN$ then TERMINATE else wait();

A5.4 Formal of algorithms for finding centers, medians, and diameters:

The formal description of the algorithm for finding the centers, medians and diameter , when single node initiates the algorithm, is given below. First we describe the notations used in this section. Each node initializes the local variables upon receiving a message that causes transition from IDLE to any other state. The various procedures such as SET_MSG, COMPUTE_MSG etc. are dependent on the application of the basic algorithm, and are described under each application.

upon $RCV(msg)_j$ = upon receiving the message from node j.

s = start node.

Variables at node i:

N(i) = set of neighbors of node i.

h_1, h_2, pd = local vars. initialized to 0.

$state_i$ = state of node i, initialized to IDLE.

SE_i = node from where the EXPLORE message was received.

RC_i = set of nodes from where the RETURN message has been received., initially nil.

Basic Algorithm SDCM:

Algorithm at node i = s:

<1>. $state_i$ = WAIT

if |N(i)| = 1 **then**

begin

SET_MSG; {sets the parameters for RETURN message}

send(RETURN) to the node j ∈ N(i).

end

else

send(EXPLORE) to all nodes k∈ N(i);

Algorithm at node i:

<1>. upon $RCV(EXPLORE)_j$,

<1a>. **if** node i in IDLE state **then**

begin

$state_i$ = WAIT;

if |N(i)| = 1 **then** { leaf node }

begin

SET_MSG;

send(RETURN) to the node j.

end

else

begin

SE_i = j;

send(EXPLORE) to all nodes k∈N(i)-{j}.

end

end

<2>. upon $RCV(RETURN)_j$,

```
<2a>.  COMPUTE_MSG;  RC_i = RC_i ∪ {j};
       if state_i = IDLE then
          begin
            SE_i = j;
            if |N(i)| = 1 then     { leaf node }
             begin
                state_i = TERMINAL;
                RESOLVE;
             end
            else
             begin
                state_i = WAIT;
                if |N(i)| = 2 then
                begin
                      SET_MSG;
                      send(RETURN) to N(i)-{j};
                end
                else
                      send(EXPLORE) to all k∈N(i)-{j};

            end
          end

<2b>  if state_i = WAIT then
```

```
begin
    if |N(i)|-|RC_i| = 1 then
     begin
        SET_MSG;
        send(RETURN) to SE_i;
     end
    else
     if |N(i)|-|RC_i| = 0 then
        begin
            state_i = TERMINAL;
            RESOLVE;
        end
end
```

A1.1 Diameter finding algorithm - SDIA

The RETURN message consists of two parameters, p_1 and p_2. Following computations are performed at node i.

INIT

```
begin
    h_1=h_2= pd = 0;
end
```

COMPUTE_MSG:

begin

p_1=p_1 + 1;

set $h_1$andh_2 as the two largest p_1's computed so far.

pd = max(pd, p_2, h_1+h_2);

end

The procedure SET_MSG sets the parameters p_1 and p_2 for sending a RETURN message.

SET_MSG:

begin

p_1 = max(h_1,h_2);

p_2 = max(pd, h_1+h_2);

end

RESOLVE:

begin

if i = s then set pd as the diameter and EXIT;

else

begin

pd = max(pd, h_1+h_2);

send(SETDIA(pd)) to node SE_i;

end

end

Each node i upon receiving SETDIA(pd) message, performs the following computations.

i. if i = s, then it sets diameter as pd and exits the algorithm.

ii.otherwise, sends the SETDIA(pd) message to SE_i.

A1.2 Center finding algorithm SCEN:

The RETURN message consists of one parameter namely p_1.

INIT:

begin

h_1=$h_2 = 0$;

end

COMPUTE_MSG:

begin

compute $h_1$andh_2 as the two largest p_1's received so far,

from nodes $max\,1$ and $max\,2$.($h_1 \geq h_2$).

end

SET_MSG:

begin

$p_1 = h_1 + 1$;

end

RESOLVE:

begin

if h_1=h_2 **then**

begin

node i is the center and exits the algorithm.

end

else

if h_1–h_2 = 1 **then**

begin

node i is the center;

$p_1 = h_2 + 1$;

SEND(RETURN) to *max* 1;

EXIT;

end

else

begin

$p_1 = h_2 + 1$;

send(RETURN) to node *max* 1;

end

end

A1.3 Median finding algorithm - SMED:

The RETURN message consists of two parameters, $p_1$andp_2.

INIT:

```
 begin
      lnum = lsum = stnum = stsum = 0; stnod = nil;
 end

COMPUTE_MSG:
 begin
      lsum = lsum + p1 + p2;
      lnum = lnum + p2;
 if p2 > stnum then
   begin
            stnum = p2;
            stsum = p1;
            stnod = j;
   end
 end

SET_MSG:
 begin
      p2 = lnum + 1;
      p1 = lsum;
 end

RESOLVE:
 begin
      Δ = lnum + 1 - 2 stnum;
```

```
if Δ ≥ 0 then
  begin
    node i is the median;
    p1 = lsum - stnum - stsum;
    p2 = lnum - stnum + 1;
    send(RETURN) to stnod; {find another median, if any}
    EXIT;  /*exit the algorithm*/
  end

else
  begin {median lies along a path in T(stnode,i)}
    p1 = lsum - stnum - stsum;
    p2 = lnum - stnum + 1;
    send(RETURN) to stnod; {locate the median }
  end
end
```

2 Algorithm MDCM

The basic algorithm MDCM, which supports multiple node initiating the algorithm, is described formally. Applications of MDCM to determine the diameter, center and medians are not described since these descriptions are similar to that for the algorithm SDCM. Minor changes required for these applications are evident from the informal descriptions of the algorithms, given earlier. The variables at each node are the same as in appendix 1. The set S, $S \subseteq V(T)$ denotes the set of nodes that initiate the algorithm.

Basic algorithm MDCM:

Algorithm at a node $i \in S$:

```
<1>. begin
        state_i = ACTIVE;
        if |N(i)| = 1 then
         begin
             SET_MSG;
             send(RETURN) to the node k∈N(i);
         end
        else
         send(EXPLORE) to all nodes k∈N(i);
    end
```

Algorithm at node i:

<1>. Upon $RCV(EXPLORE)_j$,

<1a>. **if** node i in IDLE state **then**

begin

$state_i$ = ACTIVE;

if |N(i)| = 1 **then**

begin

SET_MSG;

send(RETURN) to node j;

end

else

send(EXPLORE) to all nodes k∈ N(i)-{j};

end

<1b>. if node i is in ACTIVE state, then the message is ignored.

<2>. Upon $RCV(RETURN)_j$,

begin

COMPUTE_MSG;

$RC_i = RC_i \cup \{j\}$;

<2a>. **if** node i in IDLE state **then**

begin

$state_i$ = ACTIVE;

if |N(i)| - $|RC_i|$ = 1 **then**

begin

```
                SET_MSG;
                send(RETURN) to the node ∈ N(i)-RC_i.
            end
        else
        if |N(i)| = |RC_i| then
            begin
                state_i = TERMINAL;
                RESOLVE;
            end
        else
                send(EXPLORE) to all nodes k∈ N(i)-{j};
    end

<2b>.  if node i in ACTIVE state then
    begin
        if |N(i)| - |RC_i| = 1 then
            begin
                SET_MSG;
                send(RETURN) to the node ∈ N(i)-RC_i;
            end
        else
        if |N(i)| = |RC_i| then
            begin
                state_i = TERMINAL;
```

RESOLVE;

end

end

end

<3>. **if a node i in TERMINAL state, it ignores all messages.**

VITA

Mohan Sharma received his B.S. and M.S. degrees from the University of Mysore, India in 1981 and 1984, respectively. In January 1987, he joined the Ph.D. program in the Department of Computer Science at Louisiana State University, Baton Rouge, Louisiana where he worked on the design and analysis of algorithms for distributed systems. His other research interests include computer networks, parallel processing, computational geometry, and robot navigation.

DOCTORAL EXAMINATION AND DISSERTATION REPORT

Candidate: Mohan B. Sharma

Major Field: Computer Science

Title of Dissertation: Designing Efficient Algorithms For Distributed Systems

Approved:

S. Vt Z
Major Professor and Chairman

F. Glen Kemby
Dean of the Graduate School

EXAMINING COMMITTEE:

Sukhamay Kak

David Huff

Doris L. Carver

J Bush Jones

Lonnie Ludwig

Date of Examination:

April 26, 1990