EFFICIENT HEURISTIC FOR MULTICASTING IN ARBITRARY NETWORKS

RAHUL KATRAGADDA

A THESIS

IN

THE DEPARTMENT

OF

COMPUTER SCIENCE AND SOFTWARE ENGINEERING

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF MASTER OF COMPUTER SCIENCE

CONCORDIA UNIVERSITY

MONTREAL, QUEBEC, CANADA

JUNE 2008

Your file Votre référence
ISBN: 978-0-494-45305-6
Our file Notre référence
ISBN: 978-0-494-45305-6

Abstract

Efficient Heuristic for Multicasting in Arbitrary Networks

Rahul Katragadda

Multicast is a communication model in which a message is sent from a source to an arbitrary number of distinct destinations. Two main parameters that are used to evaluate multicast routing are the time it takes to deliver the message to all destinations and the traffic, i.e., the total number of links involved in the multicast process. It has been proved that finding an optimal multicast solution on both time and traffic is NP-hard. We propose a heuristic for the multicasting problem in an arbitrary network. We perform extensive simulations to test our heuristic for pure random network topology and two types of N-Level Hierarchical topologies.

Acknowledgements

I express my sincere gratitude to my research supervisor, Dr Hovhannes A. Harutyunyan, for giving me an opportunity to work under his guidance. His patience and charisma have inspired me throughout the course of my learning and will continue to do so. I appreciate his patience in outlining and explaining me the multicast topological issues and also for his instructive advice that led me focus on my research work and complete the thesis in a reasonable timeframe.

Finally, I would like to extend my thanks to the faculty of the Department of Computer Science and Software Engineering, some of who instructed me in the Master's major courses that I took; for the knowledge and skills I learned from them and for providing the excellent research facilities and resources. I would also like to appreciate the assistance that I received from the staff and fellow graduate students during my studies in Concordia University.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

High performance computers are always in demand for handling tasks such as 3D animation, Image processing and Pattern recognition, Weather forecasting, etc. These are complex scientific calculations that cannot be typically done on simpler machines. Multicomputer [1], also known as MPC (Massively Parallel Computer) [8], is the preferred high performance machine in such situations. A multicomputer basically has thousands of processors inter-connected like a network with each of them having their own dedicated memory to operate on. A complex task on such a system is divided into simpler tasks and then fed to multiple processors to operate on concurrently thereby reducing the time and increasing the efficiency of the machine. However, these processors need to communicate efficiently with one another over their network, consequently defining the efficiency of the system as a whole [8].

Different communication methodologies exist such as Unicast (one-to-one),

Broadcast (one-to-all) and Multicast (one-to-many). Unicast and Broadcast are just special cases of Multicast. Inefficient multicast communication could become the bottleneck in the overall performance of the system. This leads to the requirement for highly efficient algorithms for improving the overall performance of the system. A lot of earlier research has concentrated on developing and improving multicasting algorithms that dealt with efficient transmission mechanisms over networks. However, this research was not generalized for any underlying arbitrary network topology. These algorithms function optimally only on their theoretically pre-assumed topology of the network such as Mesh, Torus, Hyper-Cube, etc. However, practical applications are different from theoretical analysis and hence, it might not be feasible to use any of such pre-defined topologies in a multi-computer network. Across all of the current research, no algorithm has been proposed for efficient multicasting over arbitrary networks except for a couple of algorithms for broadcasting [1, 5]. In this thesis, we provide an efficient multicast algorithm that could be applied in any given arbitrary network.

1.2 Problem Statement

New network service requirements and modern high capacity network techniques have created an undisputed necessity to develop more efficient methods to share information than before. Multicasting is a method of sending the same information to several recipients simultaneously. It is an enhancement to traditional unicast and broadcast transmission methods. In the following paragraphs, we will provide more information on various network environments: Unicast, Broadcast and Multicast. This information is needed to better understand the multicast topologies and algorithms that we would

present in later chapters.

In a Unicast environment, the transmitter sends the data to only one recipient. This can be typically observed in data communication networks where the recipient's network address is used to reach it. For example, the network address can be an IP address or an IPX address. If a transmitter needs to send the same information to multiple recipients, it must use several addresses in order to reach all these recipients. This causes the transmitter to send the same data several times and waste large amounts of resources.

In a Broadcast environment, the same information is sent to all recipients. The recipient can then choose if he needs this information and then possibly discard it. From the transmitter's point of view, the data is sent only once and usually to one address only. Broadcasting does not consume transmitter's resources more than single unicasting but the resources of the network infrastructure are consumed in a quite inefficient way. Apart from Radio and television transmissions, modern data communication networks are good examples of broadcasting environments. Some protocols use broadcasting to discover resources from the network. For example, it is quite usual that a workstation sends a broadcast message to a network to discover the server machine. In data communications networks, broadcasting is usually restricted to one physical or logical network segment in order to decrease the load from the whole network. If broadcasting were not restricted then broadcast message floods from all parts of the network could cause the total load to rise exponentially.

Multicasting is a method of communication that is used to reach several recipients by one transmission. Recipients must also be defined separately and it must be possible to restrict those recipients who receive the data that are sent. The result of one multicast

transmission is exactly the same as when unicast transmission is committed several times consequently—once for each recipient. The major difference is that the transmitter has to send data only once and hence making it consume as few resources as possible. The data is also sent to only those recipients who were expecting it making a very efficient use of the network's resources. Figure 1 shows the Unicast, Broadcast and multicast transmission methods.

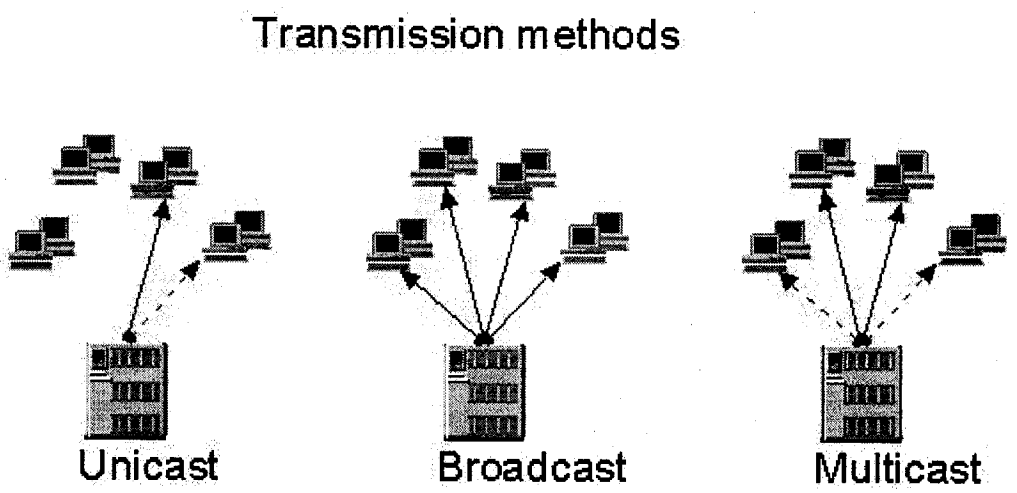


Figure 1: Different Transmission methods.

Multicast time and multicast traffic are the two parameters under consideration while implementing an optimal multicast algorithm. These parameters however are not independent of each other and hence the problem of finding an optimal solution is NP-Hard. Various multicast algorithms had been proposed with time and traffic optimization for well-defined topologies such as 2D, 3D Meshes [28, 29], Torus [9, 30] and Hypercubes [12, 31]. These algorithms require that the underlying multi-computers have their respective topologies to work ideally with their defined optimizations. When the

underlying network is arbitrarily selected like an intranet or an internet, these algorithms might have poor performance. This thesis deals with the problem by proposing a heuristic for multicasting in an arbitrary network.

1.3 Scope of the Study

In general, multicast is a huge domain considering the different network topologies being operated on and the specific applications being used. Many problems that occur in several scenarios such as traffic, dead-locks, inactive links, etc., need to be addressed [19]. Therefore, this thesis will concentrate on an ideal network topology, i.e., it is assumed no such faults occur just for the case of simplicity. We provide a solution based on an algorithm that we have developed for use over any network topology. We concentrate in this thesis on the theoretical outcome of the research.

In this thesis, the network topologies being considered can broadly be classified into two different categories—Random and Hierarchical. The sections of chapter 2 have an in-depth explanation of these topologies for better understanding of their structure and workings. Furthermore, this study only focuses on the single message multicast whereas the multiple message multicasting is completely out of scope. Also, due to the NP-Hardness of the problem, we are not concerned about the best optimal solution, but are only going to provide a medium through which we can reach that point by using some of the better ways to communicate using the multicasting model.

The simulation of the algorithm is done on a typical computer with each network end-point (or processor) being a node in a graph and the links between the network being the edges in the graph. Also, the network links are assumed to be bi-directional and the

distance between inter-connected endpoints to be of unit weight.

Any of our proposed algorithms, when used in any practical situations, might further be optimized based on different network parameters such as the underlying network architecture, or specific properties such as the throughput, degree of the network, type of network, etc.

1.4 Organization of the Thesis

In the following chapters we will first introduce some concepts related to the work being performed followed by the critical analysis of the SSH algorithm being proposed for multicasting.

In Chapter 2, we introduce the different network topologies being used for our analysis. In particular, we will introduce two different classifications of network topologies based on their structural properties.

Chapter 3 presents the algorithms that form the core of the thesis. Since this is a first work on such a topic, we are going to use a random store-and-forward multicasting algorithm called the Brute-force algorithm as a basis for comparing the results of our SSH algorithm. We define both algorithms along with their respective time-complexities.

In Chapter 4, we present our algorithm's experimental analysis, which puts out the statistical results of our simulation on paper along with comparisons for the algorithms—both graphical and numerical. We then examine how efficient our newly proposed algorithm is compared to the Brute-force algorithm. We use multicast time and

multicast traffic for calculating and defining the efficiency of our proposed multicasting algorithm.

Chapter 5 provides a conclusion of what we have done for improving the multicasting performance on arbitrary network topologies and our deductions based on our analysis. Finally, suggestions to graduates for future work based on our research are described.

Chapter 2

Literature Review

2.1 Preliminary

In this chapter we present the user with the necessary concepts to better understand this thesis.

Theoretically, any network could be always modeled as a graph. Hence any network related problem (multicast algorithm in our case) is typically converted into a graph theory problem. *A network* can be modeled as a graph G = (V, E), where V is the set of vertices (or nodes) and E is the set of edges (or communication lines). Two nodes $u \in V$ and $v \in V$ are *adjacent* if there is an edge $e \in E$, such that $e = (u, v)$. We also say node u or v is a *neighbour* of another node. The *degree* of a node is the number of neighbors of this node. The *degree* of a graph G is the maximum degree among all nodes in this graph. Δ stands for the degree of a graph. A path p in a graph G = (V, E) is a sequence of nodes of the form $p = v_1, v_2, \ldots, v_n$, $(n \geq 2)$, in which each node v_i is adjacent to the next node. Obviously, the path p is also a sequence of edges. The length of a path

is the number of edges in the path. The length of the shortest path between two nodes is the *distance* between them. The term "shortest" in this case can refer to least number of edges, least total weight, etc. The *diameter* of a graph is the maximum of the distances between all pairs of nodes in the graph. A graph G can be directed or undirected. The graph is *undirected* if the adjacency relation defined by the edges is symmetric or $E = \{\{u,v\} \mid u, v \in V\}$ (sets of vertices rather than ordered pairs), otherwise the graph is *directed.* A graph G = (V, E) is said to be *connected* if there is a path between any two nodes on G. It is obvious to assume that the network is represented by a connected graph. A graph with no path that starts and ends at the same vertex is called an *acyclic* graph.

Multicast can be defined as the problem of disseminating a piece of information, owned by a certain node called the source node, to all the destination nodes. These destination nodes form the multicast group nodes. Multicasting is performed by placing a series of calls along the communication lines of the network. At any given instance of time, the informed nodes contribute to the information dissemination process by informing one of their uninformed neighbours. In this thesis we always use a multicast model that can be described in the following steps:

1. In each call, only one informed node and one of its uninformed neighbours are involved.
2. Each call requires one unit of time.
3. A node can only participate in one call per unit of time.
4. In any given unit of time, multiple calls in parallel can be performed on different nodes.

A *multicast scheme* of an originator or source u is a set of calls that completes the multicasting in the network, originated at vertex u. An optimal multicast scheme informs all the destination nodes in the least amount of time.

Suppose there is a graph called $G(V, E)$, where each node in V corresponds to a node and each edge in E corresponds to a communication link. Also consider a set $K = \{ u_0, u_1, u_2, \ldots u_k \}$ such that in a communication process, any member u_i of K may need to multicast data to the other members $(K - \{ u_i \})$. The set K is a subset of $V(G)$, which is called a multicast group.

Multicasting on a network can be implemented using two different approaches—Unicast-based multicast and Tree-based multicast

Unicast-based Multicast

Implementing unicast-based multicast is straight forward. In unicast-based multicast, a separate copy of the message from the source node to every destination node is sent. This method is called *separate addressing* [27]. This is a simple but expensive solution because it is done sequentially and it does not use any of the previously used links.

Tree-based Multicast

In the Tree-based multicast approach, the source node sends the message to a selected set of its neighboring nodes. It will be disseminated from node to node in a certain order. This will continue until all destinations receive the message. All nodes and links that are involved in the dissemination, will occur only once at a certain time and a tree will be formed rooted at the source node. In this way, a multicast scheme can be described as a tree spanning the multicast group members.

The advantage of the tree-based multicast approach compared to unicast-based multicast is its high efficiency on both time and traffic. This means that the destination nodes will share paths in order to never use the same link twice, which will reduce the amount of total traffic. Also high parallelism in message dissemination will reduce the communication latency significantly. It should be noted that a multicast scheme can be shown by a tree.

Efficiency and scalability are considered to be the main parameters for evaluating multicast communication. A small multicast delay in disseminating a message results in a more efficient multicast. To achieve a tree with small multicast delay, the time needed to deliver a message from a source to all destinations in the tree should be calculated. In other words, the maximum time it takes for the last destination to receive the message should be considered.

It should be noted that given a source node in a tree, automatically there would be a direction from the source to the leaves for passing information. So the source can be considered as the root of the tree and the neighbours via outcomming edges can be defined as the children of a node.

Given a tree $T(V, E)$ and a source node $u \in V$, the optimum multicast time $m(u, T)$ can be calculated with a bottom-up approach using the following formulas:

1. If u is a leaf node, which means it does not have any children, $m(u, T) = 0$.
2. $M(u, T) = max\ (i + m(v_i, T_{vi}))$ in which v_i denotes the ordered set of the children of u and T_{vi} is the subtree of T rooted at v_i such that subtrees T_{vi} are labeled in a way that $m(v_i, T_{vi}) >= m(V_{i+1}, T_{vi+1})$.

Figure 2 illustrates how the message transmition delay or latency is calculated.

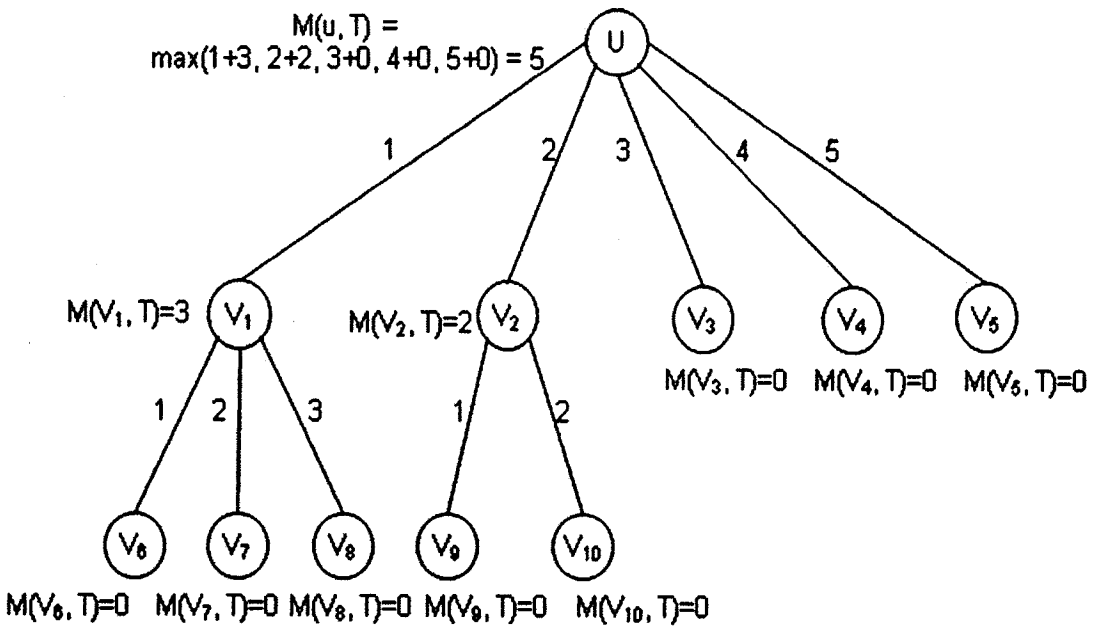


Figure 2: Calculation of optimum multicast time.
(It will take u, 5 time units to disseminate a message to all the child nodes.)

2.2 General Network Topologies

A network is represented as a collection of nodes inter-connected by links [25]. The network topology is determined only by the configuration of connections between nodes—hence making it a part of graph theory. Although in real life, a physical network might be affected by the properties of its components such as the distances between nodes, physical interconnections, transmission rates, and/or signal types, they are not a matter of concern in our study of network topologies. The nodes are inter-connected using a variety of topologies that can be broadly classified into two categories—Direct and Indirect [16].

2.2.1 Types of Network Topology

In direct networks, each node has a point-to-point or direct connection to some of the other nodes, called neighboring nodes. Examples of direct network topologies include ring, star, mesh, tree and many more as shown in Figure 3 (Source: [25])

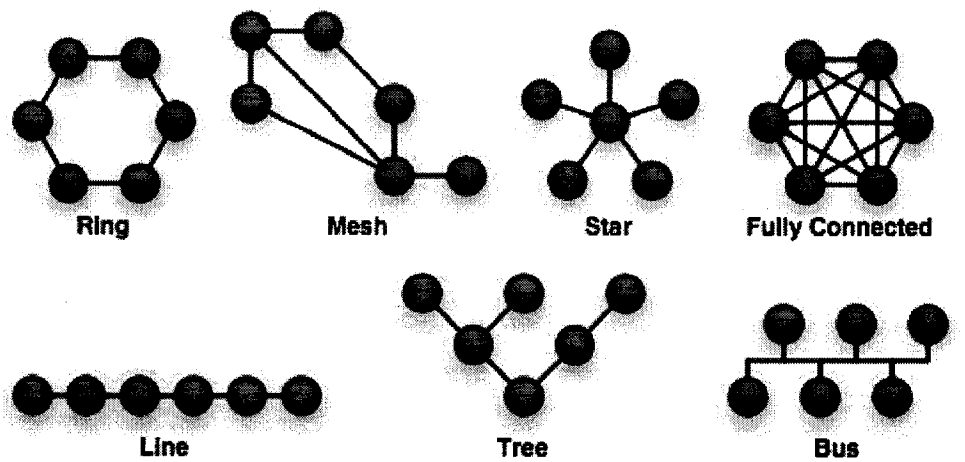


Figure 3: Common Network Topologies

In direct networks, if there are at least two nodes with two or more paths between them, then it is called a mesh network. In indirect networks, the nodes are connected to other nodes or a shared memory through one or more switching elements. Examples of indirect networks include crossbar, bus, and multistage interconnection networks. Among them, the bus network is most commonly seen. A bus network is a network architecture in which clients are connected via a shared communications line, called a bus [25]. There are several common instances of the bus architecture, including one in the motherboard of most computers, and those in some versions of Ethernet networks. Bus networks are the simplest way to connect multiple clients, but often have problems when two clients

want to communicate at the same time on the same bus. The advantages of a bus network are that it is easy and cheap to implement, it is easy to set up and change (add or remove node) and failure of one node does not affect the others. The disadvantages of a bus network are the length of the bus and the number of stations being very limited. Failure of the bus will disable the whole network and performance degrades quickly as more nodes are added because bus contention becomes the bottleneck of the network.

Mesh networks, on the other hand, are more reliable and fault tolerant. Failure of one node or several links will not cause the whole network to fail. Another good characteristic of a mesh network is that it scales well. Mesh networks have hence emerged as a popular architecture for massively parallel computers (MPC). To make it even easier to scale and manage in MPC, mesh nodes are usually arranged as an n-dimensional array and nodes are connected to their neighbors in each dimension, consequently called n-dimensional mesh. Here we present formal definitions for n-dimensional mesh networks.

Definition 2.1.1 (n-dimensional mesh) An n-dimensional mesh is an interconnection structure that has $k_0 \times k_1 \times \ldots \times k_{n-1}$ nodes, where k_i denotes the number of nodes in the i^{th} dimension. Each node in the mesh is identified by an n-coordinate vector $(x_0, x_1, \ldots, x_{n-1})$, where $0 \leq x_i \leq k_i - 1$. Two nodes $(x_0, x_1, \ldots, x_{n-1})$ and $(y_0, y_1, \ldots, y_{n-1})$ are connected if and only if there exists an i such that $x_i = (y_i \pm 1)$, and $x_j = y_j$ for all $j \neq i$. Thus the number of neighbors of a node ranges from n to $2n$, depending on its location in the mesh [16].

We notice that, in a mesh, there is no connection between the first node and the last node of each dimension. So communication between them needs to travel a long distance along the dimension and it is also unidirectional. That's why a mesh network is

un-symmetric and has a large diameter. But if we add a link between the first node and last node in each dimension, then the line along each dimension forms a circle that will significantly reduce the travel distance (almost by half) between nodes. Because now in any dimension, a message could travel along two opposite direction and reach the same destination, the mesh therefore becomes symmetrical and we call the mesh with wrap-around links a Torus. The following is the formal definition of an n-dimensional torus [8].

Definition 2.1.2 (n-dimensional torus) A n-dimensional torus is defined as an interconnection structure of n dimensions having $k_0 \times k_1 \times \ldots \times k_{n-1}$ nodes, where k_i denotes the number of nodes in the i^{th} dimension. Each node in the n-dimensional torus is identified by an n-coordinate vector $(x_0, x_1, \ldots, x_{n-1})$, where $0 \leq x_i \leq k_i - 1$. Two nodes $(x_0, x_1, \ldots, x_{n-1})$ and $(y_0, y_1, \ldots, y_{n-1})$ are connected if and only if there exists an i such that $x_i = (y_i \pm 1) \bmod k_i$, and $x_j = y_j$ for all $j \neq i$. There are wraparound channels in the n-dimensional torus (specified by the use of modulus in the definition), which are not present in n-dimensional meshes. If $k = 2$, then every node has n neighbors, one in each dimension. If $k > 2$, then every node has $2n$ neighbors, two in each dimension.

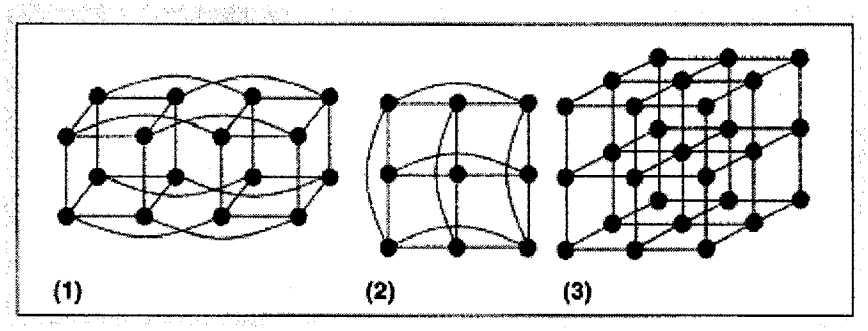
The torus is a symmetrical topology in which the degree of a node is the same irrespective of its location in the network. Thus, unlike the mesh, all the nodes in tori are identical in connectivity, which balances traffic load and simplifies the algorithm to handle them. The mesh, though, is an asymmetrical topology in which the node degree depends on its location. Interprocessor communication performance depends on the location of source and destination. The channels near the center of the mesh experience higher traffic density than those on the periphery. The network diameter of a mesh is

greater than that of the torus with the same number of nodes [16]. Hence average distance between nodes in a torus is reduced significantly over a mesh. But the pitfall of torus is that, in practice, the wiring is more difficult because of the wraparound links and wrap-up links are usually different from regular links (longer), which result in different weight of edges. Therefore, the scalability of a torus may not be as good as a mesh. Some special cases of n-dimensional meshes and torus have special names, notably *K*-ary *n* cube and hypercube. Here are their definitions.

Definition 2.1.3 (K-ary n cube) A *k*-ary *n*-cube is a special case of *n*-dimensional torus when the size of all *n* dimensions is the same, say, *K*.

Definition 2.1.4 (Hypercube) A hypercube is a special case of *n*-dimension mesh when the size of all *n* dimensions is 2. It is also a special case of *K*-ary *n* cube when $K = 2$, hence called *2*-ary *n* cube [8].

K-ary n cube and hypercube are even more symmetrical and regular structure than torus. Therefore, there are many special ways to deal with routing in such a network topology, notably the addressing of the node and calculation of the distance. They are popular topologies for first generation multicomputer. For example, n-cube or hypercube was adopted by all first-generation hypercube multicomputers and by iPSC-2 and nCube-2/3. Whereas, low-dimensional mesh and torus become more popular in new generation multicomputers include Touchstone Delta [Intel 1990], Paragon [Intel 1991] and Symult 2010 which uses 2D mesh, Cray T3D [Kessler and Schwarzmeier 1993; Scott and Thorson 1994], MIT J-Machine [Noakes, et al. 1993][8] which uses 3D mesh. Figure 4 (Source: [8]) shows a *4*-dimensional hypercube and a *3*-dimensional mesh and a *2*-dimensional Torus.



(1) 2-ary 4-cube(Hypercube); (2) 3x3 2D Torus; (3) 3x3x3 3D Mesh;
Figure 4: Mesh-connected Network Topologies

2.3 Graph Models

This section presents the different graph models that we will be using in our study and analysis. These graph models are basically classified on how they are constructed—purely randomly or hierarchically. We will be describing three different models in the following sections.

2.3.1 Flat Random Graphs

A wide variety of flat (i.e., non-hierarchical) random graphs have been used in networking literature to simulate inter-network models. All of these models are just simple variations of the standard random graph model, which is basically constructed by distributing vertices randomly on a plane. Then each pair of vertices is considered for an edge between them with a probability of α. Although, building a network in such a way does not simulate any real life networks, this network model is extensively used because of its relative simplicity in designing and for studying common network problems.

A simple change in the above method of graph generation provides different variations of pure random graph. In this case, the vertices are randomly distributed on a plane just like a pure random graph and then the probability function used to calculate the

probability of an edge between a pair of vertices is altered. These functions are modeled to better reflect the real world networks such as internet/intranet. One such very common model in the network literature is the Waxman [3] model, where the probability of an edge from vertex u to v is given by the following formula:

$$P(u, v) = \alpha\, e^{id/(\beta L)}$$

Where $0 < \alpha, \beta \leq 1$, d is the Euclidean distance from u to v and $L = \sqrt{2}\ x\ scale$ is the maximum distance between any two nodes. Increasing α will increase the number of edges in the graph, while an increase in β increases the ratio of long edges relative to shorter edges. This is called the Waxman 1 Model. Several different variations of Waxman model have been proposed with the following changes:

- Replace d by a random number between 0 and L [3]. This is the Waxman 2 model.
- Scaling $P(u, v)$ by a factor $k\,\epsilon / n$, where ϵ is the desired average node degree, n is the number of nodes and k is a constant that depends on α and β [2].
- Allow $\alpha > 1.0$ [4].

Although, both the Waxman models are very similar, the second model is a lot more interesting because of the fact that the addition of the factor $radius = k\,\epsilon$ gives more direct control over the number of edges in the graphs that are generated, provided k is known. Another variation of this model called the Doar-Leslie model, is created by choosing the α parameter of the Waxman model to be equivalent to any of the parameters k, ϵ, n and α.

Exponential model and Locality model are two other random models that are proposed with the intention of relating the edge probability to the distance between

vertices (as in the Waxman model) but with a more direct probability function. In the Exponential model, the edge probability decreases exponentially with the distance between the two vertices and hence the name. The Locality model, on the other hand, partitions the edges into discrete categories based on the length of the edges, and assigns a different edge probability for each category. Another nice feature of this model is that we can extend many of the analytical results of (pure) random graphs to this model. Also, any finite number of categories is allowed within this model, which gives us a better control over the topologies generated. The Exponential model can be expressed as:

$$P(u, v) = \alpha\, e^{-d/(L-d)}$$

The Locality model with two categories and using a parameter *radius* to define the boundary of each of the categories can be expressed as:

$$P(u, v) = \begin{cases} \alpha \text{ if } d < L \times radius \\ \beta \text{ if } d \geq L \times radius \end{cases}$$

The following is a quick run-down table showing the edge probability between two vertices at Euclidean distance d for the most common models that have been defined above. For the purpose of this study, we will only focus on the Pure Random model because of its simplicity and randomness.

Model	***Edge Probability***
Pure Random	α
Waxman 1	$\alpha\, e^{-d/(\beta L)}$
Waxman 2	$\alpha\, e^{-rand(0,L)/(\beta L)}$
Doar Leslie	$\alpha\, (radius) / ne^{-d/(\beta L)}$

Exponential	$\alpha e^{-d/(L-d)}$
Locality	α *if* $d < L$ *x radius* β *if* $d \geq L$ *x radius*

Table 1: Flat Random Graph Models

2.3.2 Hierarchical Models

The random models discussed above and the currently being presented hierarchical models represent two extreme ends in the sense that random models present very little control over the structure of the resulting topologies, whereas the hierarchical models are highly rigid in the way they are structured. Neither of them might capture the exact hierarchy that is available in real life networks. However, both reflect some notion of locality that clearly identifies if a set a nodes are more likely to be connected than the others. Real life networks can be widely classified into two types—Internet and Intranet. The Internet is simply a global network of networks. It is the connection between two or more networks that are not necessarily physically connected. The Intranet on the other hand is a private network of networks such as a corporate network only accessible to the employees of the corporation.

The following is a discussion of creating these hierarchical topologies by inter-connecting smaller components or networks together based on a pre-defined structure very similar to the way intranet or internet functions.

N-Level Hierarchical Model

The topology of an N-Level hierarchical model is constructed recursively. We first start with a connected graph. Then at each step in the recursion, each node in the

current topology is completely replaced by another connected graph. The edges in this newly created graph are 'resolved' in various ways for example using the pure random models defined above.

To construct an N-Level hierarchical model, we first divide the Euclidean plane into equal sized square sectors, whose number is determined by a parameter called Scale (*S*). Then, a flat random graph is generated with each node being placed in any one of the S^2 sectors (it is not necessary for each sector to have a node.). The top level graph in the whole topology is now constructed with a scale parameter of S_1. Recursively, each sector having a node is then sub-divided again based on a second level scale parameter S_2 and another graph is constructed (as earlier) by using this sector as a unit plane. Euclidean co-ordinates for all nodes are now adjusted to fit this new scale, in the sense that *n* sectors of the original top-level plane are now $n \times S_2$ sectors. Recursively, again the process continues until we have a final graph that is the result of the product of all the individual scales at each level. The edge lengths are then roughly determined by the edge level. For example, consider a three-level graph. Most if not all of the top-level edges (edges part of the original graph) are longer than the second-level edges which are in turn longer than the third-level edges. One advantage of the hierarchical model is that we can define the routing policy based solely on the principle that routes within a domain should stay entirely within that domain.

Figure 5 shows the graphical layout of the pure hierarchical graph. Although far from ideal, this figure represents a good network topology of a huge corporate network (intranet) with a main domain being the corporate network itself which is then divided

into sub-domains representing the different branches of the corporation, which are further subdivided into multiple departments in a branch.

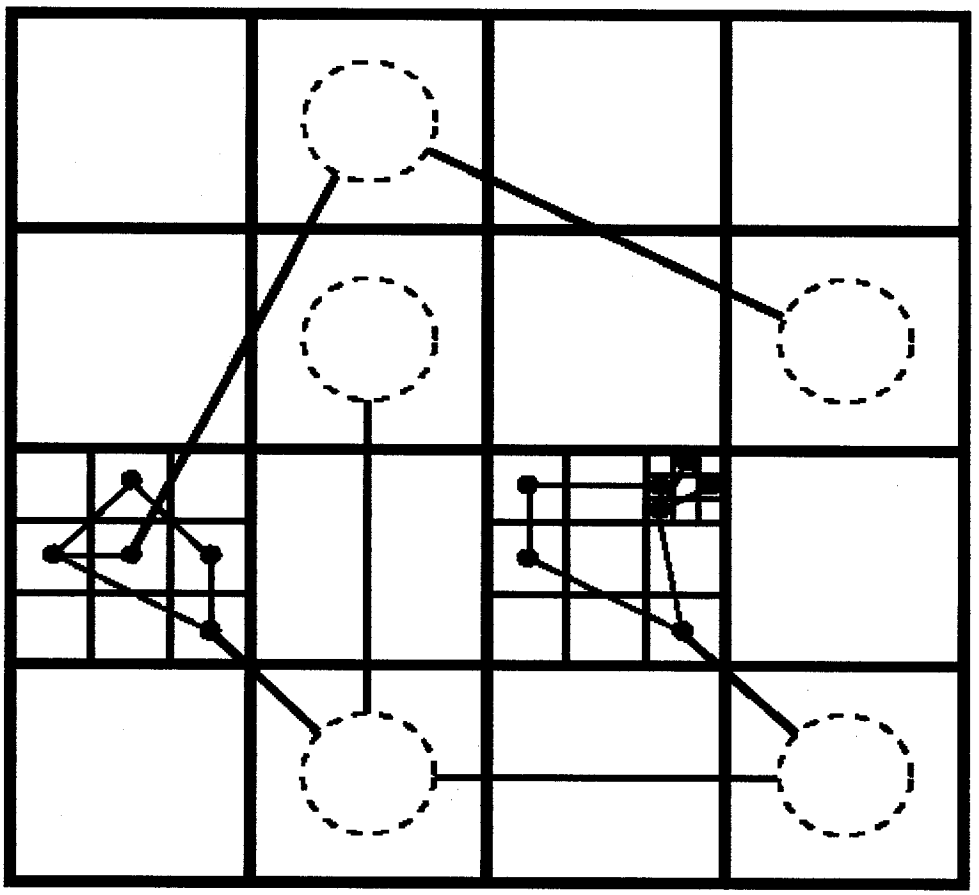


Figure 5: N-Level Hierarchical Layout

Transit-Stub Hierarchical Model

Unlike the N-level hierarchical model, the Transit-Stub model produces hierarchical graphs in a completely different way. It starts by first creating inter-connected transit and stub domains (see Figure 6). A connected random graph is first created using any of the methods described earlier. Each node in this graph represents the entire transit domain. Each of these nodes is then replaced by another connected random graph, which basically

acts as a back-bone network for that transit domain. Once all the transit domains are created, a number of connected random graphs each representing a stub domain connected to one of the nodes of the transit domains are then created. Finally, additional edges are added between pairs of nodes—one from the transit domain to one from a stub.

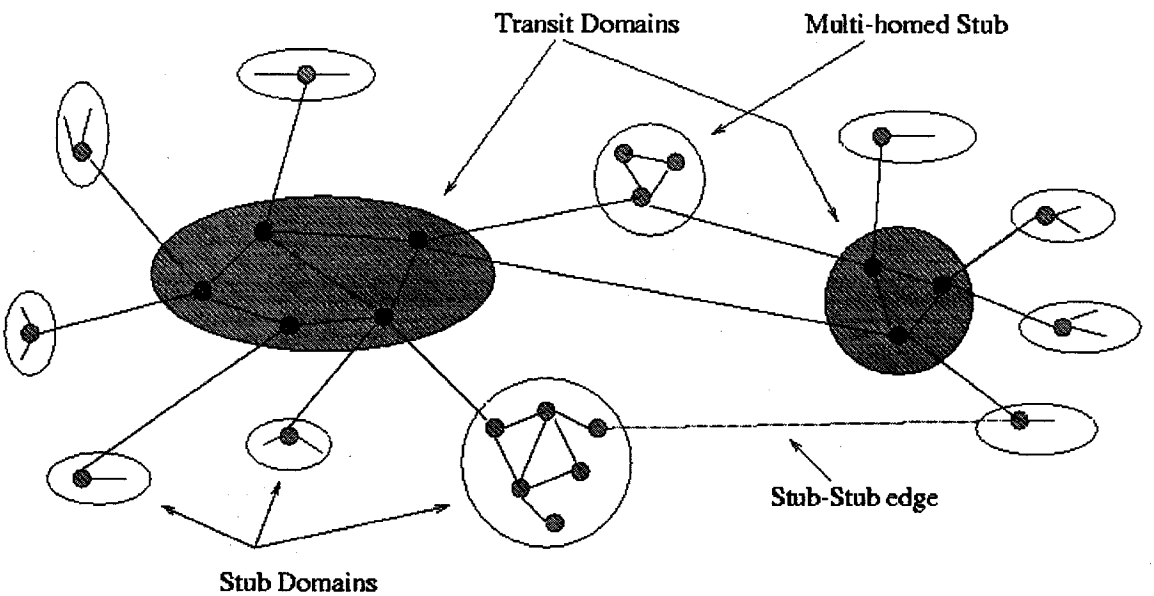


Figure 6: Transit-Stub Domain Structure

The size of the graph (number of nodes) and distribution of nodes between transit and stub domains is controlled by the parameters shown in Table 2.

The total number of nodes in a Transit-stub graph is given by the formula:

$$\text{Total \# of Nodes} = T \times Nt\ (K \times Ns + 1)$$

The example in Table 2 shows a case of how easy it is to generate rather large graphs with this method. Although, the parameters shown here (except *T*) are all average values, the number of nodes is very much randomized while keeping the average constant.

Parameter	Meaning	Example
T	# transit domains	6
$\mathbf{N_t}$	(avg) nodes/transit domain	15
K	(avg) stub domains/transit node	12
$\mathbf{N_s}$	(avg) nodes/stub domain	8
	Total Nodes	8730

Table 2: Transit-Stub graph parameters

The domains in transit-stub graphs are placed in the plane in such a way that they may overlap one another. The same scale is used throughout the construction process as depicted schematically in Figure 7. A top-level graph determines the location and connectivity of the transit domains, i.e., each node of the top-level graph corresponds to a transit domain. Each node of the top-level graph is then replaced by a complete transit domain graph, whose scale (as a fraction of the total length of a side of the unit square) is determined by the parameter '*transfrac*' of the network simulator ns-2. Stub domains are created with a scale that is some fraction of the global scale. This fraction is again a parameter '*stubfrac*'.

Transit-Stub is a well known model for the Internet. The Internet can be viewed as a set of routing domains that are merely inter-connected independent networks. All nodes in a domain share routing information. Two types of domains namely *Transit* domain and *Stub* domains exist. For any path through a domain, if atleast one of the end nodes is in the domain, it is a stub domain. Else, it is a transit domain. Just as with the Internet, interconnected transit and stub domains compose the graphs generated by the

Transit-Stub method. Moreover, this model could also produce graphs having very realistic average node degrees. Stub domains might further be classified as single homed and multi-homed stub domains. Multi-homed stubs connect to more than one transit domain, while single-homed stubs connect to only one transit domain. A transit domain is composed of a set of backbone nodes, which are fairly highly connected to each other (a backbone node has a degree of at least 2). Some stubs also have links to other stubs. The routing characteristics of the Internet can be summarized by the following general principles:

- The path connecting two nodes in a domain stays entirely within that domain.
- The shortest path connecting node u in stub domain U to node v in another stub domain V goes from U through one or more transit domains to V, and does not pass through any other stub domains.
- When two stub domains are connected directly via a stub-stub edge, the path between two nodes on the two domains may (but need not) go along the edge and avoid any transit domains.

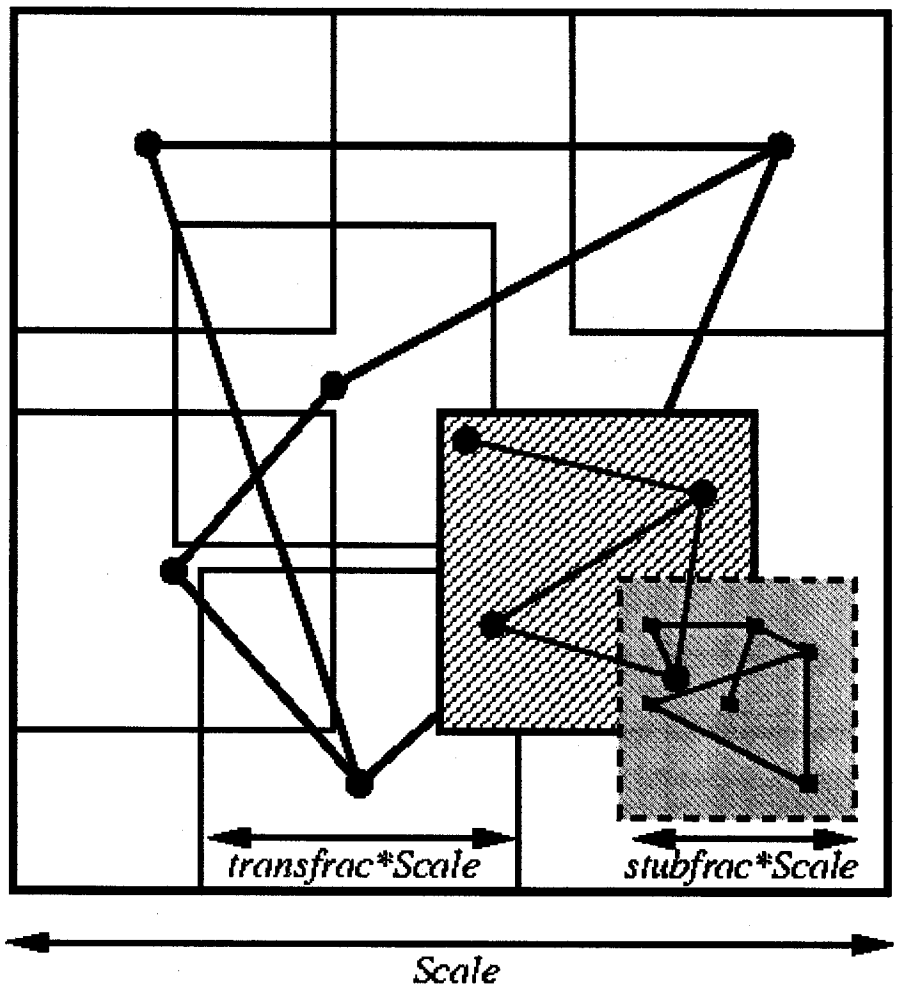


Figure 7: Transit Stub Layout

Chapter 3

Multicast Algorithms

Two different algorithms defined for multicasting are elaborated in this chapter. The first algorithm, called the 'Brute-force Algorithm', is a simple algorithm that is used to get some base results to compare with the other algorithm. The second algorithm, called the 'Simple Search Heuristic Algorithm' (SSH), aims to achieve optimal multicast time and traffic in the topologies described in chapter 2. The discussion starts by presenting the basic assumptions for proper functioning of the algorithms followed by the algorithms themselves and their time complexities.

3.1 Assumptions

Multicast performance can be evaluated based on a wide variety of criteria that typically depend on the architecture of the underlying network, such as the topology of the network, routing techniques being used, switching technology and node architecture. The combination of these criteria in different ways might result in completely different

performance evaluation. Hence, the following constraints are being assumed to make sure that the performance evaluations of the different algorithms discussed here are done on the same playing field.

- The node could use either One-port or All-port architecture. In One-port architecture, at any instant of time, a node may transmit/receive the message to/from only one of its neighboring nodes. Conversely, in an All-port architecture, a node can transmit/receive message(s) to/from several different neighboring nodes concurrently at each instant of time. The current discussion however only focuses on One-port architecture and the simulations presented in next chapter also reflect the same.
- Neighbouring nodes are connected by links that have the same length, bandwidth and transmission speed. Hence, each link is equally weighted and is considered to be of unit magnitude.
- Bi-directional links are the norm, i.e., the message could be transmitted in both directions between the two neighbouring nodes.
- A semi-distributed routing scheme is presumed to be utilized by the underlying network. In this routing mechanism, the multicast routing algorithm is executed at the source node with all the required information, which then generates the multicast tree or path. Then the message, with only information of destination node and replicate node of multicast tree embedded in header, is sent out from source node and disseminated from node to node. At each intermediate node, a very simple routing algorithm is executed to decide which nodes the message should be forwarded to according to the routing information in the header.

- The message could be replicated at any intermediate node. So at any instant of time, it is possible that multiple copies of the message are being transmitted over many different links simultaneously.
- For the purpose of this thesis, the process of multicast communication is started when the source first sends the message out and ends when all the destination nodes receive the message.
- Only one multicast message is being transmitted through the network at any given time, i.e., there is no multi-message multicasting (otherwise also called Gossiping).
- The underlying network is considered to be faultless and free of any errors that might occur in a practical world such as deadlocks, traffic control, blocking and fault tolerance. Only the theoretical aspect and structure of the network are considered.

3.2 Tree-based Multicast Algorithms

The trademark of any efficient multicast algorithm is that the transmission of a message from a source to each destination node takes considerably less time and fewer communication channels (links) compared to other algorithms. Given a set of destination nodes, the algorithm uses a common path to transmit the message as much as possible and then branches to transmit the message to the remaining destination nodes resulting in tree-like routes, hence the name Tree-based multicast. Each graph can have many such multicast tress built for a given source and set of destination nodes. However, our goal is to find the tree that is optimal in both time and traffic during the course of multicasting.

This is an NP-Hard problem and hence impractical. Heuristic algorithms are used to find a solution and the heuristics use a pro-time approach to build the multicast tree, i.e., the heuristics try to reduce the multicast time as the first priority and then if possible try to reduce the multicast traffic.

This thesis showcases two multicast algorithms, the Brute-force algorithm and the Simple Search Heuristic algorithm (SSH). The Brute-force algorithm is a basic store-and-forward algorithm that functions by randomly forwarding the message from a node to its neighbours until all the destination nodes have received the multicast message. As the name suggests, the algorithm does not use any special logic to transmit the message. Although this might be very a simplistic and inefficient way to multicast, it is presented in this thesis for comparison with the Simple Search Heuristic algorithm (SSH) presented later. The SSH algorithm, obviously uses heuristics to achieve near optimal multicast time while keeping the multicast traffic as little as possible. The following sections will formally present the algorithms along with examples to clearly show the working of these algorithms followed by a discussion about their time complexities.

3.2.1 Brute-force algorithm

The main goal of this algorithm is to optimistically and randomly forward the message to any of its neighbours and hope for the best. The algorithm uses two different sets of nodes—*SourceNodesSet* and *InformedNodesSet*. *SourceNodesSet* refers to the nodes that already have the multicast message and can transmit the message to at least one of its child nodes in the next instant of time. Initially when the algorithm starts, the *SourceNodesSet* will only contain the Source node of the algorithm. *InformedNodesSet*

contains the set of nodes that do not yet have the multicast message but will receive the message from one of the nodes in the *SourceNodesSet* in the current instant of time. *InformedNodesSet* is empty initially and also after each iteration as the contents of this set are copied over into the *SourceNodesSet.*

Algorithm Brute-force

Input: Given a graph G = (V, E). *n* is the total number of nodes, *s* is the source node and $d_1, d_2, \ldots d_k$ is the set of destination nodes.

Output: Multicast Time of the graph *m(G),* which is given by *max* $(m(d_i))$, where $i = 1$ to *k.* Multicast Traffic of the graph is the total number of links used during the execution of the algorithm.

```
1:   Add s to SourceNodesSet
2:   Foreach node in SourceNodesSet
     {
3:       Select a child node Cn that has not been informed yet.
4:       Add Cn to InformedNodesSet.
5:       If node has all of its children informed then
         {
6:           Remove node from SourceNodesSet.
         }
     }
7:   If d1, d2,.... dk have been informed then
     {
8:       Goto Step 12.
     }
9:   Else
     {
10:      update SourceNodesSet with InformedNodesSet nodes.
11:      Goto Step 2.
     }
12:  Update statistical data MulticastTime, MulticastTraffic.
```

Time Complexity

As discussed, the algorithm selects a random child node and forwards the message to it. At worst, the algorithm would have to forward the message to each and every node of the

graph. Hence, the time complexity of the algorithm is O(n) where n is the number of nodes in the graph.

Example

Figure 8 shows the graph model that will be used to demonstrate the working of the algorithms proposed in this chapter. The model is an actual transit-stub graph generated by using ns-2 for a total of 20 nodes. The source and destination nodes are also randomly selected for demonstration purposes. The source node *l* is denoted by a double rounded border and the three destination nodes *e*, *h* and *t* are denoted by dashed rounded border as can be seen from the figure.

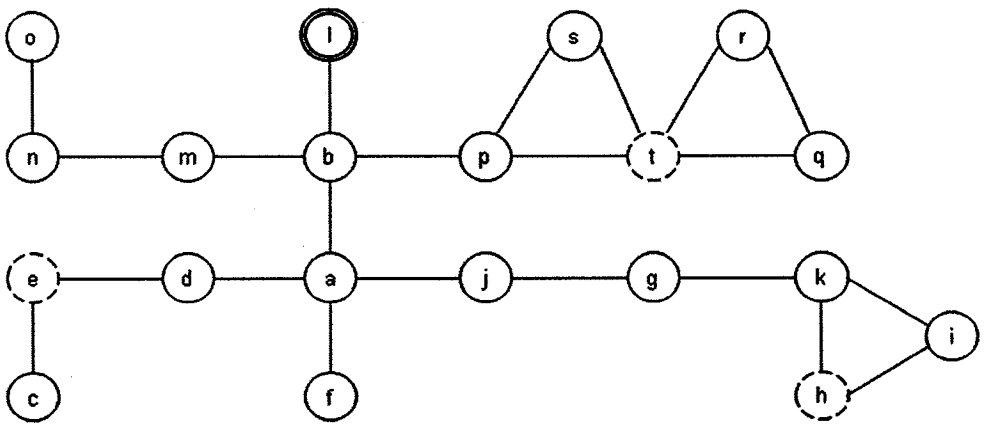
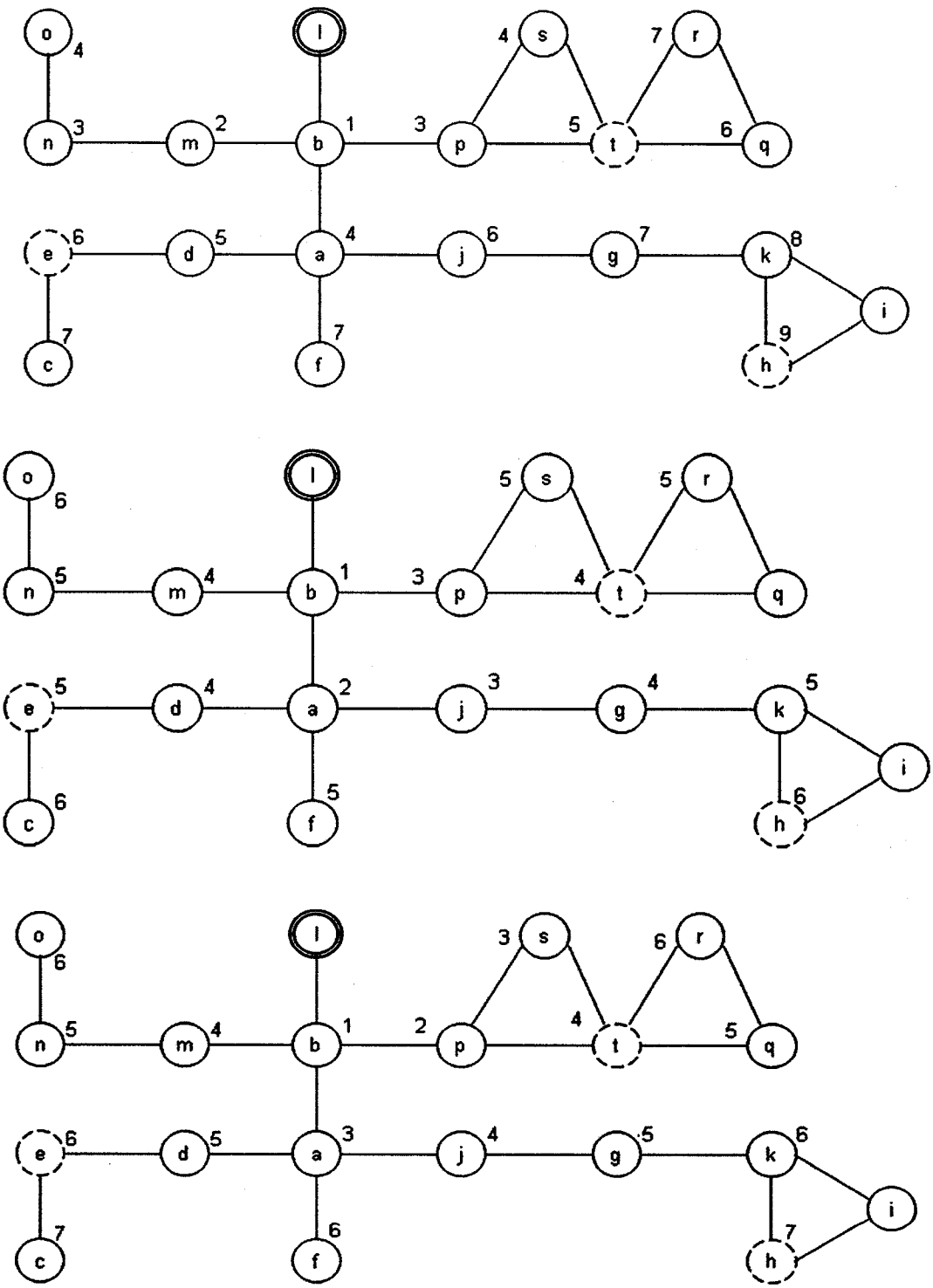


Figure 8: Example graph of Transit-Stub model with 20 nodes

The following discussion refers to Figure 8(a). The algorithm starts by initializing the *SourceNodeSet* with the source node *l*. It then randomly selects any one of its child nodes. The node *l* only has a single child node *b* and hence selects it and transmits the message to it at time 1. Node *b* is then added to the *InformedNodeSet*. Node *l* has no other un-informed child nodes and hence is deleted from the *SourceNodeSet*. Since none of the destinations have received the message, the

SourceNodeSet is updated with the contents of *InformedNodeSet* (in this case – *b*). Repeating the process, a random child node of *b* (either *a* or *m* or *p*) is selected and then sent the message in the second instant of time. For the purpose of this example, consider it to be *m,* which is then added to the *InformedNodeSet.* Since the multicasting is not complete, the *SourceNodeSet* is again updated by adding *m* to the list. The node *b* then selects its child node *p* to transmit the message at instance 3. Also the node *m* selects its only child node *n* to transmit the message. *InformedNodeSet* is updated with *n* and *p*. Since *m* has no other un-informed child nodes, it is removed from *SourceNodeSet* and updated with the contents of *InformedNodeSet.* The process is repeated until all the destination nodes have received the message. The multicast time is then given by the maximum time taken by a destination node within the graph to receive the message.

Figure 9 demonstrates three probable results of the Brute-force algorithm when executed on the graph model shown in Figure 7. Figure 9(a) has the worst multicasting time of 9, while the Figure 9(b) has the best multicast time of 6 and Figure 9(c) has an average multicasting time of 7 comparatively. These are simple approximations of how the Brute-force algorithm might perform given the circumstances. Because of the randomness inherent within the algorithm, we execute the algorithm ten times on the same set of source and destination nodes and then take the average as the multicast time for comparison purposes in the next chapter. In this case, we could take the average of the three executions and compute the average multicast time of the Brute-force algorithm to be ((9 + 7 + 6) / 3) = 7.33. As can be seen from the figures, the Brute-force algorithm generates a lot of un-necessary traffic (almost equal to Broadcasting) by transmitting the



(a)Worst Performance (b) Best Performance (c) Average Performance
Figure 9: Multiple executions of Brute-force algorithm

message to nodes that do not help with multicasting. Hence, even with the best multicasting time, the Brute-force algorithm has very bad multicast traffic performance.

3.2.2 Simple Search Heuristic Algorithm (SSH)

The goal of this algorithm is to build a near optimal Multicast Tree. The algorithm starts by first building a Breadth First search graph (Dijkstra's graph in case of directed links) until all the destination nodes are found. This graph is then trimmed by removing the un-necessary nodes and vertices. Some of the destination nodes might still have multiple paths to reach them. Then heuristics for each node on this graph are calculated to estimate the near optimal time for multicasting. Using these heuristics, we can eliminate other nodes and/or edges that are not necessary for multicasting. This process essentially builds the Multicast tree. Multicasting in this tree is then straight forward using the calculated heuristics. The heuristics that are applied on the BFS graph are modifications of those proposed by BinShao in his thesis [1].

Algorithm Simple Search Heuristic

The algorithm basically uses four different modules, with each one of them performing an important step in Multicasting. *Build BFS Graph* module takes as input the network graph and tries to build a BFS graph from it until all the destination nodes have been found. One important characteristic of the BFS graph is that any pair of parent and child nodes in the graph will differ in depth by a value of 0 or 1 only. However, this BFS graph will have unnecessary vertices and edges that were discovered while building the graph. *Trim Edges Recursively* is a recursive module that takes care of it by removing all

the unnecessary additional nodes and edges that are part of the BFS graph, thus ensuring that the destination nodes are either leaf nodes or internal nodes of the graph. *Apply Heurstics* module takes the BSF graph generated above and applies heuristics to each and every node in this graph. Once the heuristics are applied, these heuristics guide us towards an efficient multicast tree by having better heuristics for nodes and edges that contribute to it. Finally, the multicasting tree is built by using the above heuristics in the *Build Multicast Tree* module and the rest of vertices and edges not in the tree are discarded. Any optimal Broadcasting algorithm can then be used to get optimal values for multicasting in this tree and the statistical data for comparisons can be retrieved. The algorithm with a call to each of these modules is presented below followed by the algorithm for each of these modules.

Input: Given a graph G = (V, E). n is the total number of nodes, s is the source node and $d_1, d_2, \ldots d_k$ is the set of destination nodes.

Output: Multicast Time of the graph $m(G)$, which is given by $max\ (m(d_i))$, where i =1 to k. Multicast Traffic of the graph is the total number of links that are part of the multicast tree after it has been built.

1: *Build BFS Graph.*
2: *Trim Edges Recursively* to remove unnecessary edges.
3: *Apply Heuristics* to the Graph built above.
4: *Build Multicast Tree* using Heuristics.
5: Multicast and obtain *MulticastTime, MulticastTraffic.*

***Algorithm Build BFS Graph* module**

This module functions by using three different lists. *QueueList* is a list of nodes that functions as a First-In-First-Out queue. It contains all the nodes that are yet to be examined to build the graph. *VisitedList* contains all the nodes that are already visited in the BFS graph built. Although this module basically works as a Breadth First search algorithm, it stops once all the destination nodes are in the *VisitedList.* This gives us a

sub-graph of the original graph with the required destination nodes. The leaf nodes of the graph being built are stored in a list called *LeafList*. The module also uses a variable called *HasValidChild* that assists in keeping track of the nodes in *LeafList*.

Input: Given a graph G = (V, E). *n* is the total number of nodes, *s* is the source node and $d_1, d_2, \ldots d_k$ is the set of destination nodes.

Output: A trimmed Breadth First Search graph $G_{BFS} = (v, e)$, where $v \in V$ and $e \in E$ with all the $d_1, d_2, \ldots d_k$ nodes either being leaf nodes or internal nodes of the graph.

```
1:   Add s to QueueList and LeafList.
2:   While QueueList not empty
     {
3:       Remove node n from QueueList.
4:       Add n to VisitedList.
5:       Set HasValidChild = false
6:       Foreach (child node cn of node n)
         {
7:           If (cn not in VisitedList) then
             {
8:               Add cn to QueueList.
9:               Add cn to LeafList.
10:              Set HasValidChild = true
             }
         }
11:      If (HasValidChild == true) then
         {
12:          Remove n from LeafList.
         }
13:      If VisitedList contains d1, d2,.... dk then
         {
14:          return.
         }
     }
```

***Algorithm Trim Edges Recursively* module**

The subgraph generated in the previous module contains other unnecessary nodes or edges, which were discovered during the process. This module removes them by starting with the leaf nodes and then recursively removing the parent nodes (given by the variable *Parent* of the node) and edges between them until it either reaches a destination node or

the source node. The input to this module is a list of leaf nodes in the graph built. This list is populated by the *Build BFS Graph* module while buiding the graph.

Input: Given the *LeafList.*

Output: The graph $G_{BFS} = (v, e)$ where $v \in V$ and $e \in E$ trimmed off un-necessary vertices and edges.

```
1:  Select a node n from the LeafList.
2:  If (n != s) and (n not destination node) then
    {
3:      Remove n from G_BFS.
4:      Set n = n.Parent.
5:      Goto Step 2.
    }
6:  If LeafList not empty then
    {
7:      Goto Step 1.
    }
```

***Algorithm Apply Heuristics* module**

This module is the most important part of the whole algorithm and hence will be described in detail with a graphical example. The heuristic algorithm [1] calculates the broadcasting heuristic for the BFS graph built in the earlier step. Thus, it helps us to build a multicast tree by eliminating the unnecessary edges while multicasting. It does so by assigning better heuristics to nodes and vertices that are optimal for multicasting and hence would lay the foundation for optimal multicast time and multicast traffic. It works by dividing the graph into different regions based on the fact that at any given instance of time *t*, there are three different types of nodes—Informed nodes, un-informed nodes and nodes that have adjacent nodes that are informed. At time *t*, the *dark region,* which is denoted by DR(*t*), is a sub-graph of G that is composed of all uninformed nodes. The nodes in DR(*t*) that have informed neighbours compose the *dark border* region, which is denoted by db(*t*). The third region, the *bright border* bb(*t*), is composed of those

informed nodes that have uninformed neighbours. Figure 10 shows a pictorial representation of these concepts.

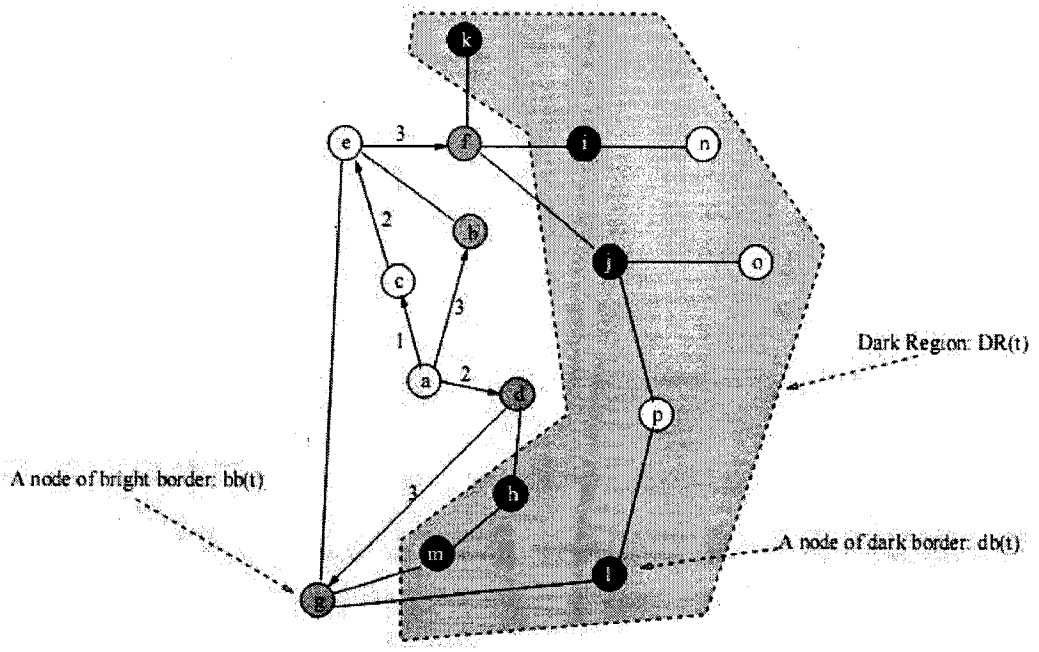


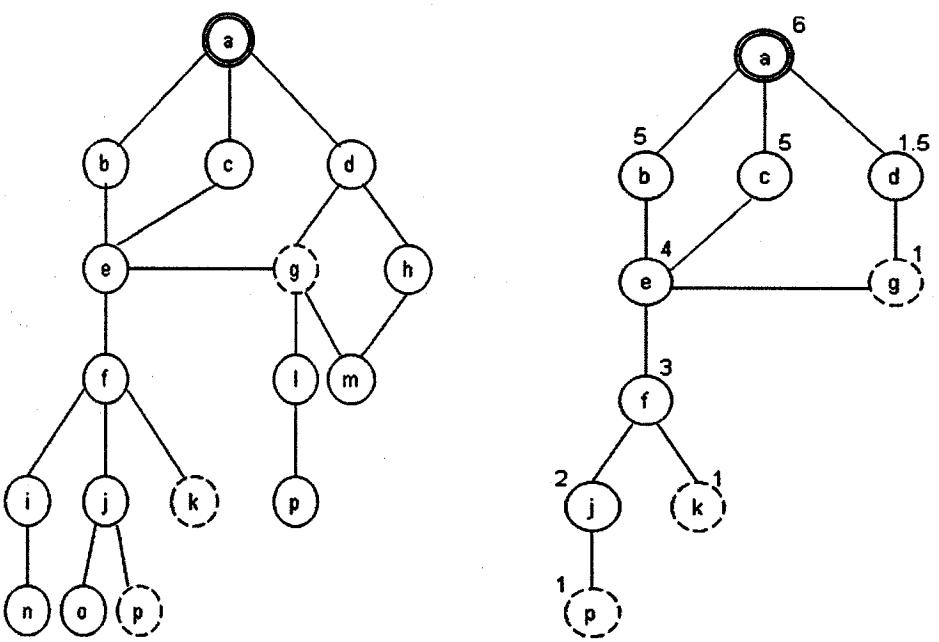
Figure 10: Several concepts in the new heuristic

In Figure 10, the shadowed area is the dark region DR(t). The nodes in DR(t) with black backgrounds belong to db(t) and the nodes not in DR(t) with shadowed backgrounds belong to bb(t). For any uninformed node *v*, *D(v, t)* denotes the shortest distance from *v* to a node in bb(*t*) at round t. Given an uninformed node *u* and its uninformed neighbor *v*, if D(*u, t*) = D(*v, t*) + 1, we say *u* is a child of *v*. Node *v* and all its descendants make up the descendant graph of *v*, which is denoted by DG(V, E, *v*) or simply DG(*v*). The multicast or broadcast time of DG(*v*) in round t is denoted by EB(*v, t*) or simply EB. It is calculated as follows.

- At round t, if a node *u* has no children, then EB(*u, t*) = 1.

- Otherwise, given u has p children, and c_1, c_2, .. c_p are the children of u ordered so that $EB(c_j, t) \geq EB(c_{j+1}, t)$, for $1 \leq j \leq p - 1$, then $EB(u, t) = \max \{ (EB(c_i, t) / n) + i \}$ for $1 \leq i \leq p$ and n is the number of parents of c_i.

The *maximum-number-weight* matching between set db(t) and bb(t) at round t, denoted by mnw(t) can be defined as follows. Only nodes in db(t) have weight. The weight of a node in db(t) is assigned as EB of the node. The number of node pairs where one is in db(t) and the other is in bb(t) is maximized. In this case, the sum of weights in db(t) is maximal.



(a) BFS Graph (b) Trimmed and Heuristics Applied to BFS Graph

Figure 11: Heuristic application procedure

The mnw(t) is assigned to nodes rather than edges. In round t, the mnw(t) can be calculated by the following process:

1. In bb(t), select the node v with minimum number of uninformed neighbours.

2. Among the uninformed neighbours of v, get the node u with maximum weight. Then, u is the mate of v.
3. Mark u as informed and repeat from step 1.

Figure 11(a) shows the BFS graph built for the network shown in Figure 9 with node a being the source and nodes p, k and g being the destinations. Figure 11(b) shows the graph after the execution of the algorithm till the end of this module with the respective heuristics of each node.

Input: Given the BFS graph $G_{BFS} = (v, e)$ where $v \in V$ and $e \in E$.

Output: The graph G_{BFS} with the Heuristics applied to each and every node in the graph.

1: **Initialize** bb(t) so that bb(0) has one node: the source s.
2: **Calcualte** EB(u, t) as the weight to any node u in DR(t).
3: **Find** the mnw(t) between bb(t) and db(t) and during the process mark all matched nodes as informed.
4: **Compute** bb(t + 1).
5: **If** bb(t + 1) is empty, the process is complete, and t would be the broadcast time. Otherwise, go to (2).

Algorithm *Build Multicast Tree* Module

This module uses two different lists to build the multicast tree. *QueueList* serves the same function as it did in the *Build BFS Graph* module above. *MarkedList* is used to store all the nodes that are marked as part of Multicast tree. Each node keeps track of the multicast path to it by updating its *Path* variable. The module works by using the heuristic value of the child node to select the best optimal path for each source node at a given instant of time. However, it can only select one child node or path for each source node. Hence, this algorithm works like an actual multicast algorithm while building the tree. All the destination nodes d_1, d_2,.... d_k will have the optimal multicast tree path stored in their *Path* variable by the end of this module.

Input: Given the G_{BFS} with heuristics applied to each node in the graph.

Output: The optimal multicast tree T.

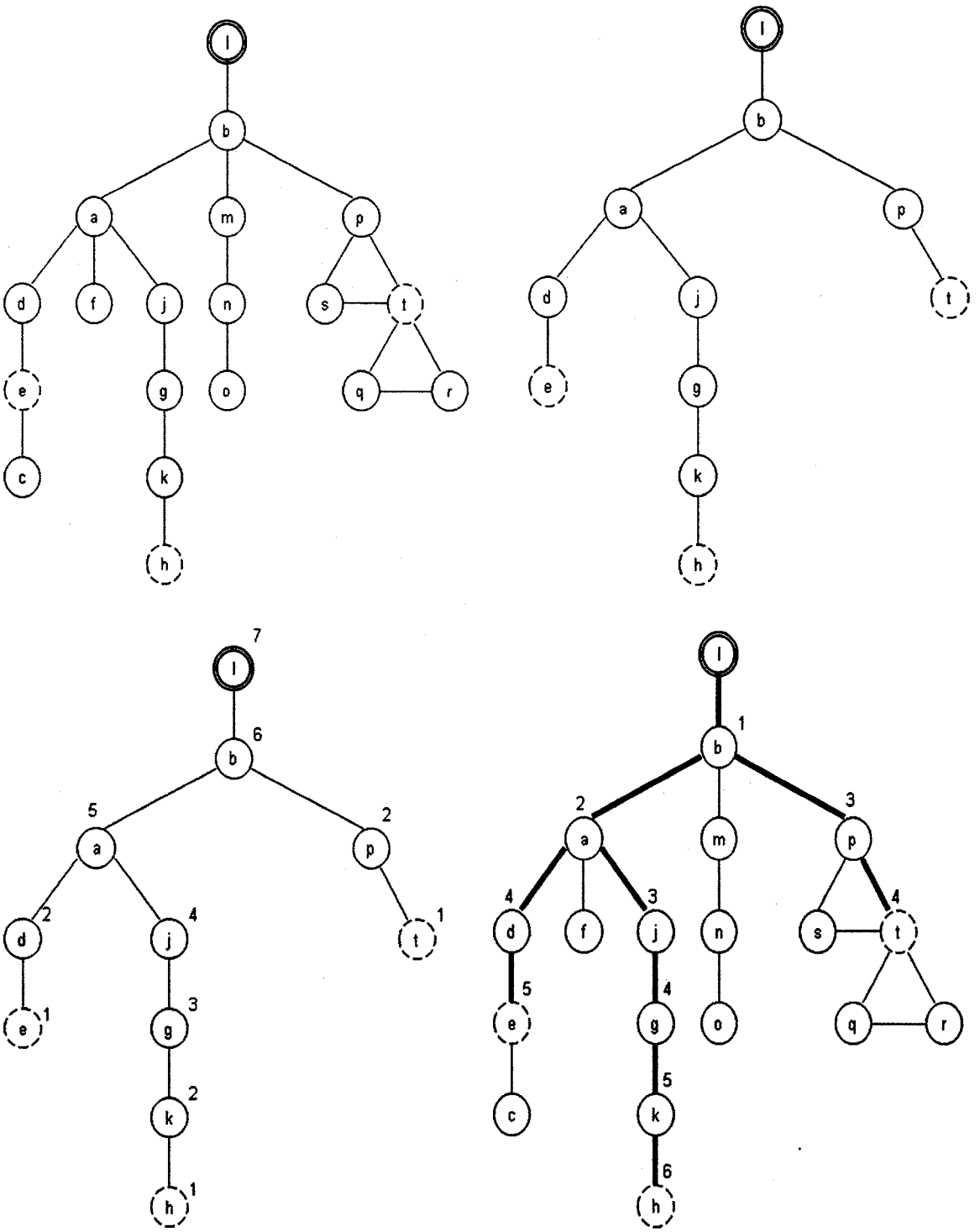
```
1:   Add s to QueueList and MarkedList.
2:   While QueueList not empty
     {
3:       Remove node n from QueueList.
4:       If Exists (child node cn of n
                        where HeuuristicValue(cn) is maximum
                        and cn Not in MarkedList)
         {
5:           Set cn.Path = n.Path + cn .
6:           Add cn to QueueList and MarkedList.
         }
7:       Else
         {
8:           Remove n from QueueList.
         }
9:       If MarkedList contains d1, d2,.... dk then
         {
10:          return.
         }
     }
```

Time Complexity

The time complexity of the algorithm is equal to the sum of Time complexity of building the multicast graph, trimming it of unnecessary edges, applying the heuristics and building the multicast tree. The time complexity of building the BFS graph can be given by O(n) where *n* is the total number of nodes in the graph. The time complexity of trimming the leaf nodes module is always negligible compared to any other module and hence can be ignored. The complexity of the heuristic algorithm is given by O(R.m) where m is the number of edges in the BFS graph and R is the number of rounds of broadcasting [1]. Hence, the Time complexity of the algorithm is O(n + R.m) or for large number of nodes O(n).

Example

The graph model of Figure 8 is used here again to demonstrate the algorithm. The Simple Search Heuristic algorithm starts with the *Build BSF Graph* module. It takes the graph provided and builds a Breadth First Search graph until all the destination nodes are found. The BFS graph built would look as in Figure 12(a). *LeafList* will now contain all the leaf nodes of the graph, i.e., *c*, *f*, *h*, *o*, *s*, *q* and *r*. This BFS graph is then trimmed of all the un-necessary edges that do not lead to the destination nodes. This is accomplished by the *Trim Edges Recursively* module by using the list of leaf nodes provided. The resultant graph would be as shown in Figure 12(b). This graph is then considered by *Calcualte Heuristics* module for evaluating the heuristics. Heuristic calculation starts from the leaf nodes. It then moves upwards towards the source node as shown in Figure 12(c). Each node is marked with their respective heuristics. The final module *Build Multicast Tree* uses the heuristics to create an optimal multicast tree as shown in Figure 12(d). This example (being a hierarchical model) does not reflect much difference between the Figures 12(c) and 12(d). However, in a large network or a purely random graph model, each node might have multiple paths to receive messages from the source node (Figure 11 with node *e* having multiple paths from source node *a*). The additional paths that are not optimal or are redundant with an existing path are removed in the final step when the module *Build Multicast Tree* is executed. The optimal multicast time of this graph is always equal to the heuristic value of the source node minus 1. The multicast time of this example using SSH algorithm is 6. Also, as can seen from the figures, the SSH algorithm uses only those nodes and edges that are absolutely necessary for multicasting thus reducing the multicast traffic generated during multicasting.



(a) BFS Graph (b) Trimmed BFS Graph (c) Heuristics Applied (d) Multicast Tree

Figure 12: Execution of SSH Algorithm

Chapter 4

Simulations and Discussions

4.1 Simulation Model

To evaluate the performance of these new algorithms and verify whether the design goal has been achieved, software to exactly simulate the multicast communications in different multicomputer networks has been developed. This simulation model and software are discussed first. Then an analysis of the performance of these new multicast algorithms is done with the help of performance curves, statistics and comparisons.

The goal of this research can be expressed as a hypothesis [32] that should be tested. A Hypothesis is a tentative theory or supposition that explains the behavior we want to test. The Hypothesis in our case can be formally defined as follows:

Hypothesis 1: Multicasting using the SSH algorithm produces efficient Multicast time and Multicast traffic compared to the Brute-force algorithm.

4.1.1 Model of Simulation Program

The goal of designing the algorithm was to make it work in a very generic manner such that the code could be re-used later for other multicasting simulation algorithms on any network topology or for any kind of multicomputer node architecture. Hence, the object-oriented design methodology which has been long known for reusability of code and flexibility in design has been chosen for the purpose. This critical decision helped a lot in managing the code when fine tuning the algorithm was needed for optimal performance. All the classes that make up the core of the simulation program are shown in Figure 13.

In the class diagram, the solid line represents a "HAS-A" relationship between the two classes whereas the dotted line represents a "USES-A" relationship between them. *Main* is the entry point of the program, which initiates all the memory. It also presents the user with a nice graphical interface to help perform different functionality (such as Execution, Result Display, etc.). The classes *Node*, *Edge* are the basis upon which any network topology graph to be used by this program is built. The *TreeNode* class, which has a *Node* object, is basically just an encapsulation of the *Node* class and contains additional information regarding the node's neighbours such as its parents, children, etc. The *Graph* is used to represent the whole network topology using the TreeNode objects. It is very similar to building a linked list of objects in a computer language such as C++. *Edge* is a simple representation of a link between the two nodes. *EdgeCollection*, as the name suggests is a collection of the *Edge* objects that are connected to a Node or the Graph based on where it is being referred at. *NetworkAlg* is the class where the core functionality of the multicasting algorithm resides. It performs all the functions such as building the network graph, building the multicast tree (if required), then assigning the

Class Diagram

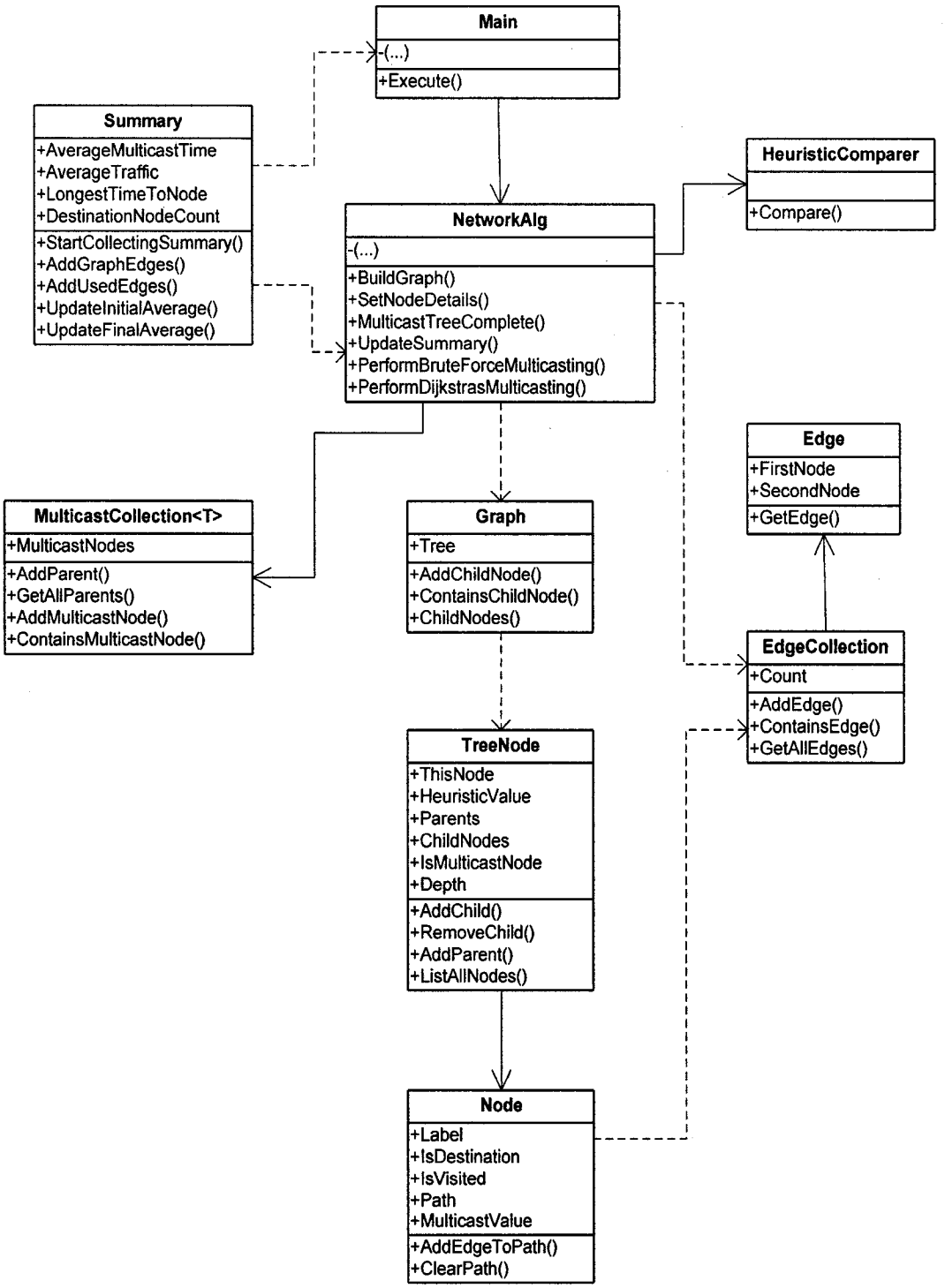


Figure 13: Class Diagram of Simulation Program

heuristic values and then multicasting based on the heuristic values. It also contains the methods for each of the different algorithms that we simulate in our thesis. *HeuristicComparer* is a special class that is used to compare the heuristic values between siblings of a node, so that a better decision over the multicasting route to be taken can be made. *MulticastCollection<T>* is a generic class that stores a collection of data of any type. The symbol '<T>' in the name of the class denotes that. Hence, it can be used to store any collection of data depending on the requirement. In this case, we use it to store a collection of *TreeNode* objects while multicasting based on the multicast time of the algorithm. *Summary* class is a static class that is used to collect statistical data while algorithms are being executed and then used to analyze this data. This data can then either be presented to the screen or saved to a file based on the user's requirement.

4.1.2 Implementation of Simulation System

The simulation program introduced earlier could be implemented in any object oriented programming language such as *C++*, Java or .Net. Since the simulation of multicast algorithm in a multicomputer network is very computationally intensive, we choose .Net, which is exceptionally good at handling collections of data very efficiently as a programming language. Also, the additional advantage of it being a managed execution environment, which basically means that we do not need to worry about any persistent application specific data in memory after the application has exited, gives us a more relaxed approach while implementing the model. More specifically, the simulation software was developed using *Visual Studio 2005* under *Windows XP* system. So, typically the program could run in any Windows operating system with .Net 2.0 libraries

installed. The computer could be any *PC*, workstation or server that a Windows operating system supports. To ensure the speed and efficiency of the simulation, we recommend a computer system with at least 2Ghz *CPU* clock and *1GB* memory.

4.2 Performance Evaluation Model

As we learned from chapter 2, there are basically two widely used parameters in evaluating the performance of multicast operation—multicast time and multicast traffic. Multicast time reflects the duration of a multicast operation from the time the message is sent out from the source node till it reaches the last destination node. Multicast traffic reflects the total number of links involved in the multicast operation. These multicast routing algorithms are all heuristic algorithms and none of them are optimal, i.e., no algorithm is best all the time, one algorithm might perform better than the other in some cases, but worse in other cases. So the average value of a number of randomly generated multicasts over multiple graphs (of similar topology) is used to reflect the performance of each algorithm.

Different types of charts based on different performance evaluation parameters are used to compare and analyze the performance of the multicast algorithms simulated in this thesis. The three charts are the *multicast time* curve, *multicast traffic* curve and *Min-Max multicast time* curve resulting from the simulations of the multicast algorithms in the three different graph models described in chapter 2. The *multicast time* curve reflects the changing trend of the multicast time with respect to the number of destination nodes. The *multicast traffic* curve reflects the changing trend of the average multicast traffic with respect to the number of destination nodes. The *Min-Max multicast time* curve reflects the

minimum and maximum multicast time of an algorithm over all of its executions on a particular network topology with respect to the number of destination nodes.

The *multicast time* and *multicast traffic* parameters are identical to the parameter *mean,* which indicates the performance of an algorithm in a certain mesh in general. This method is derived from the ideas introduced in paper [18] for evaluating the efficiency and resource consumption of multicast in an arbitrary network.

Definition 4.1.1 (Mean of Parameter) Given a graph G and a multicast algorithm A, the average value of a parameter P from all the multicast samples generated by algorithm A, regardless the number of its destinations, in a statistic experiment is called the *mean* of parameter P for multicast algorithm A in a graph G.

We use *mean* to indicate the overall performance of algorithm A on parameter P in the graph G. It roughly or vaguely says the performance of a multicast algorithm in a graph in general but not resulting from any specific case. So in our thesis research, we will have *multicast time mean* and *multicast traffic mean*, which reflect the multicast time and traffic of a certain multicast algorithm in a given graph in general. Technically the mean of a parameter is actually the average line of the whole curve of the parameter.

To compare the performance of multiple algorithms in a chart, a curve for each algorithm is drawn resulting from exactly the same set of samples. Every point on the curve is the *mean* of over 7500 runs of the multicasting algorithm with the same number and set of destination nodes. The next section explains this process in detail.

4.2.1 Simulation Methodology

The following steps enumerate the methodology followed to get the *mean* values for comparison and analysis of all the algorithms described in chapter 3. The same process is used to calculate the *mean* for both the multicast time and multicast traffic.

- 15 different graphs (5 randomizations of 3 different input values into ns-2) for a given number of total nodes are randomly generated.
- For each of these graphs, 10 different nodes are randomly selected as the source nodes for multicasting.
- Each source node is then assigned 50 different sets of multicast group nodes (destinations). These destination nodes are randomly selected for each source node. Hence the total number of source and destination sets is equal to 50 * 10 = 500.
- The multicasting algorithm is then executed on each of the above created source and destination set nodes.
- Each statistical data-point on the graph hence represents 15 * 500 = 7500 random simulations of the algorithm.
- An *initial average* is obtained by simulating the algorithm on the same graph with the same source node but using the 50 different sets of multicast group nodes. This process is repeated for all the 10 different source nodes and the average taken—say *Intermediate Average*. The above two averages are then calculated for all the 15 different graphs and the average of averages obtained—say *Final average*. This *Final average* represents the *mean* multicasting value shown in the graphs.
- An exception for the Brute-Force algorithm however is that because of the randomness of the algorithm, the multicasting on the same set of source and destination

nodes is performed multiple times and then the *mean* is used as the result of multicasting. The current simulation performs this step 10 times and then using the mean. Hence, for the Brute-Force algorithm each data-point will represent 15 * (500 * 10) = 75000 random simulations of the algorithm.

Formally, let us define a multicast that delivers message from source u_0 to k destinations $d_1, d_2, \ldots, d_{k-1}, d_k$ in graph G, the resulting multicast tree is denoted as $MT(V,E)$, where V is the set of nodes in the tree and E is the set of edges in the tree, the length of the path from the source node u_0 to destination nodes $d_1, d_2, \ldots, d_{k-1}, d_k$ is $l_1, l_2, \ldots, l_{k-1}, l_k$ respectively. The total time units waited at branching nodes on path to d_i *is* w_i ($1 \leq i \leq k$). The multicast time is defined as the maximum time units needed to deliver the message to all destinations. Basically, it is the length of the path plus the number of time units waited at replication (branching) nodes along the path. Hence we have:

$$\textit{Multicast Time} = \max \{ l_1+w_1, l_2+w_2, \ldots, l_{k-1}+w_{k-1}, l_k+w_k \}$$

$$\textit{Multicast Traffic} = | E(MT) |$$

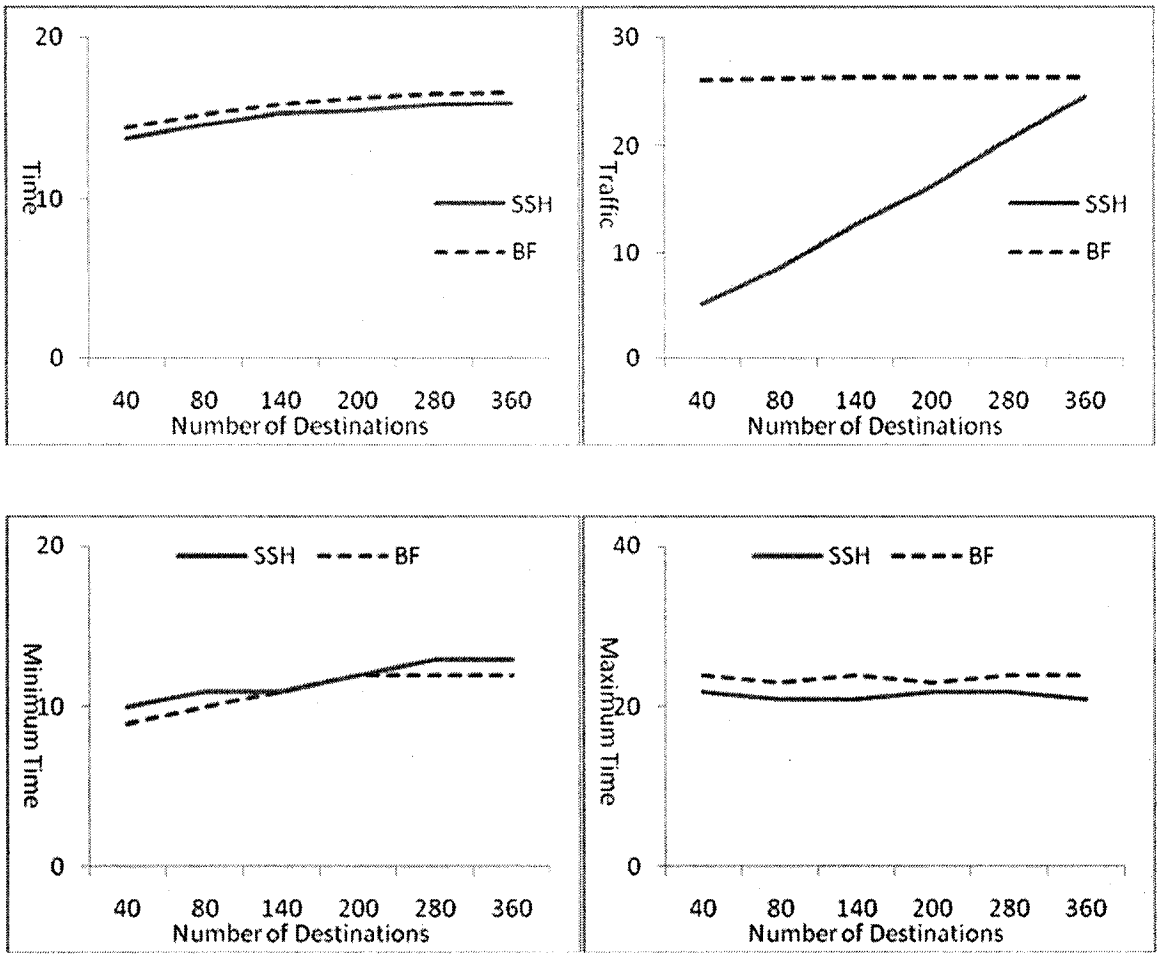
We also use the semi-distributed routing scheme for our simulation of the SSH algorithm. In this scheme, the simulation session is divided into two phases. Phase one is to execute the multicast algorithm at the source node with the given set of destination nodes. The output of phase one is the multicast tree. Phase two is to simulate the transmission of an arbitrary message. As we know that we use the number of hops to measure multicast time, so the size of the message doesn't affect the result. In our case, the multicast tree is embedded at the head of the message. To save the space, only information of all destination nodes and replication nodes organized as a tree will be stored in the head, the passerby nodes will not be included since they could be

determined through the routing between destination nodes and/or replicate nodes. Upon the time of receiving the message, a passerby node simply stores and then forwards the message to next node on the route to next destination node in the tree. But a destination node or replication node needs to first accept the message and strip its node information from the embedded multicast tree, and then break the tree into several subtrees rooted at each of its child nodes, and then send a copy of the message to each child node with its correspondent multicast sub-tree embedded.

The confidence level of all of the simulation values presented in the next section is very high. In other words, the margin of error that the algorithm would generate a different value for each data point in the statistics presented is very low. In case of multicast time and multicast traffic the margin of error is ± 0.1 and in the case of Min-Max multicast time, it is ± 2.

4.2.2 Simulation of Brute-force and SSH in Pure Random Model

Figure 14 below shows the performance of Brute-force algorithm and SSH algorithm in purely generated random graphs through simulations done on a graph containing a total of 400 nodes. The Figure 14(a) represents the multicast time curve, 14(b) represents the multicast traffic curve and 14(c) and (d) representing the min-max multicast time curves. In each of the figures, the dotted line represents the performance of Brute-force algorithm whereas the normal line represents the performance of SSH algorithm.



(a) Multicast Time curve (b) Multicast Traffic curve
(c) Min multicast time curve (d) Max multicast time curve
Figure 14: Multicast performance in pure random graphs.

In Figure 14(a), the multicast time curves for both the Brute-force algorithm and the SSH algorithm almost overlap. By virtue of the pure random model, all the nodes are highly connected with each node having multiple shortest paths to transmit messages, which cause the Brute-force algorithm to have near optimal multicast time. The multicast time mean of SSH in a pure random graph is 15.14 which is 4% less than the mean of the Brute-force algorithm, which is 15.79.

Figure 14(b) shows the amount of traffic generated through the algorithms. It is pretty clear that the SSH algorithm generates much less traffic compared to the Brute-

force algorithm. It is also noticeable that the traffic generated by the Brute-force algorithm is almost constant while that of the SSH algorithm steeply rises as the number of destination nodes increase. This is in accordance with the fact that as the number of destination nodes increase, the algorithms perform more like a broadcasting algorithm rather than a multicasting algorithm. The multicast traffic mean in this case for the SSH algorithm is 14.59 which is 44% better than Brute-force algorithms 26.20. However, at its best the SSH algorithm has an efficiency of 80% when the number of destination nodes is 40 (10% of the total number of nodes). This clearly shows that the SSH algorithm performs better with respect to the multicast traffic generated in a pure random model compared to that of the multicast time.

Figure 14(c) and 14(d) represent the minimum and maximum values of multicast time respectively during the execution of the algorithm over all the randomly generated graphs with different sets of source and destination nodes. As is the case with the multicast time shown in Figure 14(a), the performance of the algorithms in this case is also very similar and overlapping in some instances as can be seen from the figures. Specifically, when the number of destination nodes are 140 or 200, the minimum multicast time for both the SSH algorithm and Brute-force algorithm overlap with the Brute-force algorithm, which performs slightly better in other instances. For the SSH algorithm, the minimum multicast mean time is 11.67 and the maximum multicast mean time is 21.5. The Brute-force algorithm has values of 11 and 23.67 for the minimum and maximum multicast time values respectively. The Brute-force algorithm has slightly better minimum multicast time performance of 6% over the SSH algorithm. The SSH

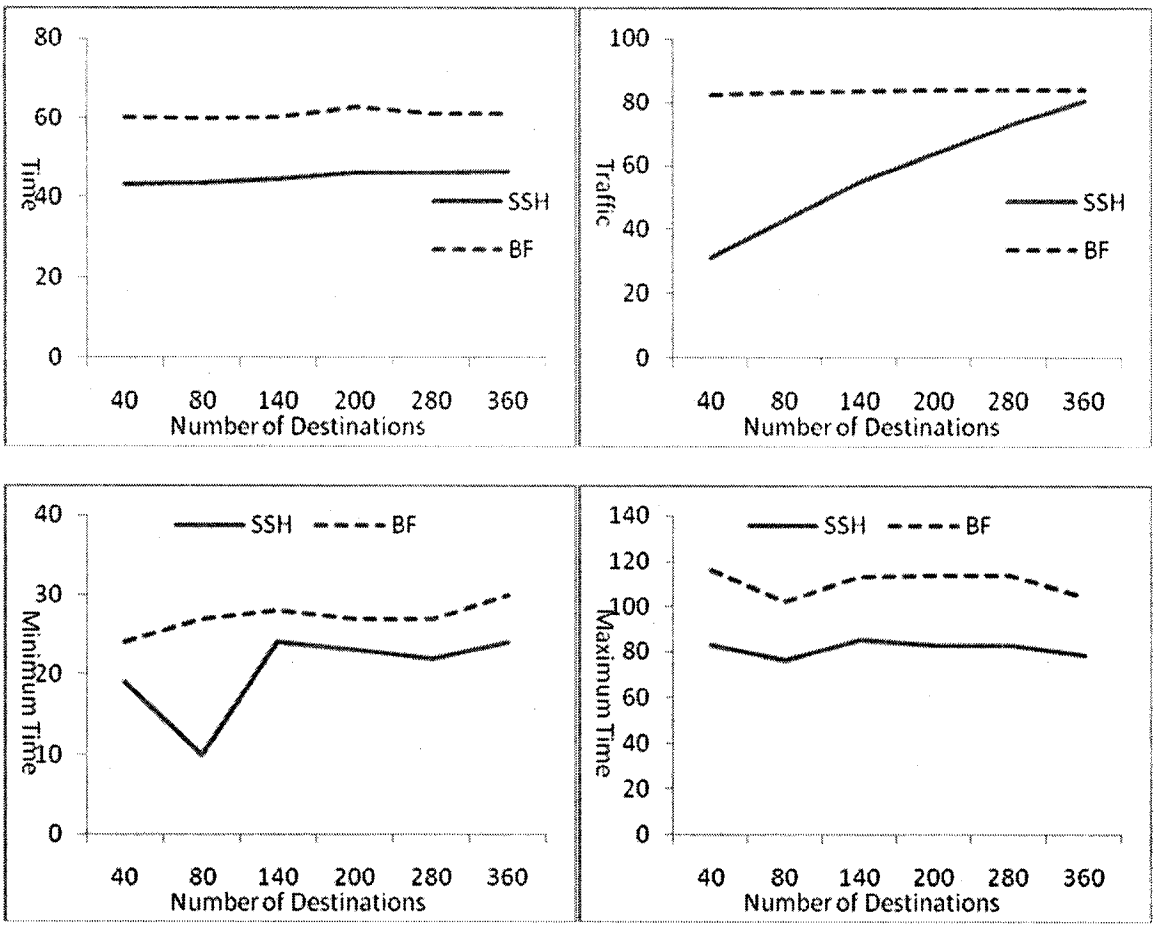
algorithm on the other hand has better maximum multicast time performance of 9% over the Brute-force algorithm.

In summary, although both the algorithms perform optimally with respect to multicast time, the SSH algorithm has a huge advantage of generating much less traffic compared to the Brute-force algorithm and hence being more efficient in using the network resources optimally. These results are even more pronounced when the number of destination nodes is comparatively much less than that of the total number of nodes in the graph.

4.2.3 Simulation of Brute-force and SSH in Hierarchical Model

Figure 15 shows the performance of SSH and Brute-force algorithms in the Hierarchical graph model, which represents the Intranet model. Again, the dotted line represents the performance of the Brute-force algorithm, while the normal line represents that of the SSH algorithm. Also the figures 15(a), 15(b) represent the multicast time and multicast traffic performance while the figures 15(c) and 15(d) represent the min-max multicast time values for the algorithms.

The multicast time of the SSH algorithm is less compared to that of the Brute-force algorithm in the Hierarchical graph model as seen in Figure 15(a). The mean of the multicast time for SSH algorithm is 45.04 while that of the Brute-force algorithm is 61.04. Hence, the SSH algorithm performs 26% better with respect to the multicast time of Brute-force algorithm. Also, as shown in the graph, the curves are almost flat indicating that there is not much difference in the multicast timing even as the number of destinations increase.



(a) Multicast Time curve (b) Multicast Traffic curve
(c) Min multicast time curve (d) Max multicast time curve
Figure 15: Multicast performance in hierarchical graphs.

Figure 15(b) is very much similar to Figure 14(b). The Brute-force algorithm generates constant traffic without much variation as the number of destination nodes increase. However, the SSH algorithm routes the messages with higher efficiency when the number of destination nodes is very minute (10% in this case) compared to that of the total number of nodes. The multicast traffic mean of SSH algorithm in hierarchical graph is 57.68, which is 31% better than the Brute-force algorithm's mean value of 83.52. Also, when the number of destination nodes is 10% of the total number of nodes, the

efficiency of SSH over Brute-force algorithm is 62%, which clearly indicates that SSH algorithm has a huge advantage in multicast traffic.

Figures 15(c) and 15(d) represent the minimum and maximum multicast time curves. The SSH algorithm clearly performs better than the Brute-force algorithm in both cases. However, as seen in Figure 15(c), when the number of destination nodes is 80 (20% of the total number of nodes) the SSH algorithm appears a have a very low value of minimum multicast time compared to its values when the number of destination nodes is less or more than 80. The maximum multicast time curves for both the algorithms seem to follow the same pattern, rising and falling at the same number of destination nodes in both cases, as can be seen in 15(d). The mean values of minimum multicast time are 20.33 and 27.17 for SSH and Brute-force algorithms respectively indicating a 25% better performance for SSH algorithm. However, when the destination nodes are 80, this performance increases to 63%. The maximum multicast time mean values for SSH and Brute-force algorithms are 81.33 and 110.5 with SSH showing a performance increase of 26%.

In conclusion, the SSH algorithm has near optimal multicast time and multicast traffic performance in Hierarchical graph model with a mean of approximately 25% improvement in case of multicast time and 30% improvement in case of multicast traffic.

4.2.4 Simulation of Brute-force and SSH in Transit-Stub Model

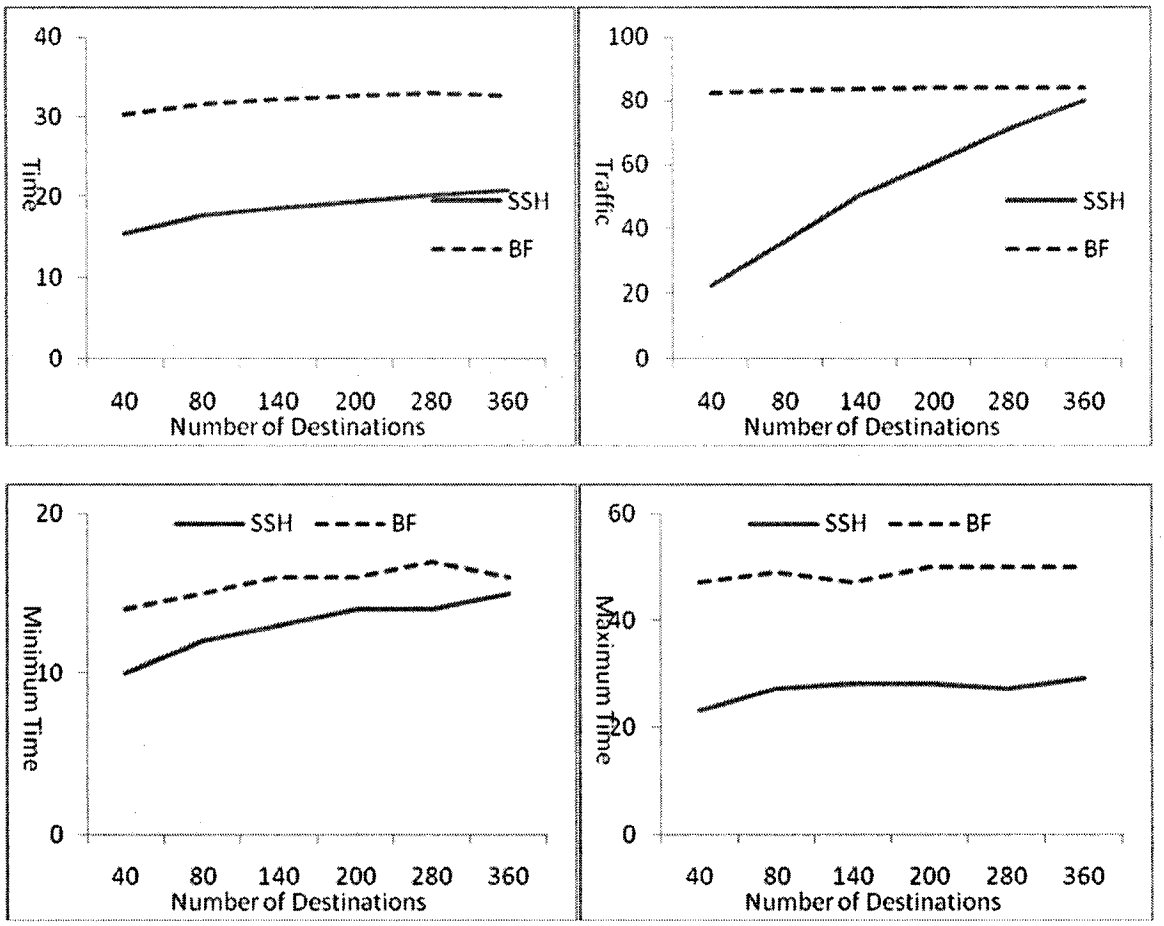
Figure 16 shows the performance of SSH and Brute-force algorithm in a Transit-stub graph model, which showcases the Internet model of the real world as discussed in chapter 2. Similar to the graphs in the previous sections, 16(a) represents the multicast

time curve, 16(b) the multicast traffic curve, 16(c) the minimum multicast time curve and 16(d) the maximum multicast time curve.

Figure 16(a) clearly indicates that the SSH algorithm outperforms the Brute-force algorithm in case of multicast time. The mean values for SSH and Brute-force algorithms are 18.67 and 32.04 with SSH algorithm having a clear performance gain of 42% over the Brute-force algorithm.

Similar to the traffic curves shown in the previous sections for pure random graph model and the hierarchical mode, the traffic curve for transit stub model shown in Figure 16(b) has a steep curve for the SSH algorithm with increasing traffic as the number of destination nodes increases and the Brute-force algorithm having an almost flat line curve irrespective of the number of destination nodes. Again, the multicast traffic mean in this case is 53.47 and 83.62 for SSH and Brute-force algorithms respectively with SSH having a 40% multicast traffic efficiency.

The minimum and maximum multicast times generated over all the iterations of algorithm executions over the graph model are shown in figures 16(c) and 16(d). The Brute-force algorithm performance in case of minimum multicast time is near optimal compared to that of SSH algorithm. SSH has a performance efficiency of 17% with the mean values being 13 and 15.67 for SSH and Brute-force algorithms respectively. However, the performance difference is greater in the case of maximum multicast time. The mean values in this case are 27 and 48.83 with SSH algorithm having a better performance of 45% over the Brute-force algorithm.



(a) Multicast Time curve (b) Multicast Traffic curve
(c) Min multicast time curve (d) Max multicast time curve
Figure 16: Multicast performance in transit-stub graphs.

SSH algorithm, as is the case in the pure random graph model and hierarchical model, outperforms the Brute-force algorithm in all cases on a Transit-stub graph model. Although it has good performance benefit in the multicast time area, it un-arguably has a huge performance benefit in the multicast traffic section. Hypothesis 1 as stated in the beginning of this chapter has hence been proven.

4.3 Confidence Intervals

The purpose of taking a random sample from a population and computing a statistic, such as a mean from the data, is to approximate the mean of the population. How well this sample statistic estimates the underlying population value is always an issue. A Confidence Interval [25, 33] addresses this issue because it provides a range of values which are likely to contain the population parameter of interest. In other words, a confidence interval is a range of values used to estimate the true value of a population parameter. They are constructed at a confidence level, such as 95%, selected by the user. This means that if the same population is sampled on numerous occasions and interval estimates are made on each occasion, the resulting intervals would bracket the true population parameter in approximately 95% of the cases.

Confidence intervals are normally represented as $(1 - \alpha)$ 100% where $(1 - \alpha)$ is called the *Confidence Coefficient* for the interval. The confidence interval for the mean of a normal population is given by the formula below:

$$Y \pm Z_{\alpha/2} (\sigma / \sqrt{N})$$

where Y is the sample mean, $Z_{\alpha/2}$ is the upper critical value of the standard normal distribution which is found in the table of standard normal distribution, σ is the known population standard deviation and N is the sample size. Some of the common confidence levels used for statistical analysis and the respective $Z_{\alpha/2}$ values are shown in Table 3.

Using the formula above, we establish a 99% confidence interval for the mean multicasting time and traffic parameters in our algorithms. Table 4 and Table 5 show the

99% confidence intervals of all the data points presented in comparison graphs earlier in this chapter for multicasting time and traffic respectively.

Confidence Level	*Z value*
80%	1.28
90%	1.645
95%	1.96
98%	2.33
99%	2.58
99.8%	3.08
99.9%	3.27

Table 3: Common Confidence Levels and $Z_{\alpha/2}$ Values

Network Model	Graph Model 1	Graph Model 2
Pure Random Model	geo 5 400 150 3 0.019	geo 5 400 240 3 0.02
Hierarchical Model	hier 5 1 5 0 5 10 3 0.3 2 4 3 0.5 5 5 3 0.2 4 8 3 0.25 2 5 3 0.3	hier 5 5 0 5 10 3 0.3 4 5 3 0.2 2 4 3 0.5 5 10 3 0.3 2 5 3 0.4
Transit-stub Model	ts 5 3 7 7 3 2 5 3 0.3 4 5 3 0.2 7 15 3 0.15	ts 5 3 2 5 4 5 3 0.3 4 7 3 0.15 8 2 3 0.2

Table 4: Input for ns-2 to Generate Network Models

Tables 5, 6, 7 and 8 show the 99% confidence intervals of two instances of each of the three networks models presented in this thesis. Tables 5 and 7 represent the confidence intervals for multicast time and Tables 6 and 8 represent the confidence intervals for multicast traffic. These graphs are generated using the network simulator ns-2 using different inputs as shown in Table 4. Each graph is executed over a fixed

source node and 50 different sets of destination nodes (or samples). However, because of the randomness inherent to the Brute-force algorithm, each graph has been executed 10 times over the same source and destination nodes. Hence, the total number of samples in this case is 500.

Pure Random Graph								
Destination Nodes	Standard Deviation		Mean Multicast Time		99% Confidence Interval			
	SSH	Brute-force	SSH	Brute-force	SSH		Brute-force	
40	1.29	1.44	12.82	13.04	12.35	13.29	12.87	13.21
80	1.38	1.55	14.92	13.54	14.42	15.42	13.36	13.72
140	1.79	1.92	17.26	15.02	16.61	17.91	14.8	15.24
200	0.93	1.28	16.36	16.54	16.02	16.7	16.39	16.69
280	0.35	1.21	14.14	14.4	14.01	14.27	14.26	14.54
360	0.4	1.52	16.86	14.84	16.71	17.01	14.66	15.02
Hierarchical Graph								
Destination Nodes	Standard Deviation		Mean Multicast Time		99% Confidence Interval			
	SSH	Brute-force	SSH	Brute-force	SSH		Brute-force	
40	1.63	2.77	44.7	53.28	44.11	45.29	52.96	53.6
80	2.31	4.44	46.5	58.92	45.66	47.34	58.41	59.43
140	1.4	4.55	50.82	62.42	50.31	51.33	61.9	62.94
200	1.17	1.96	41.52	47.84	41.09	41.95	47.61	48.07
280	0.65	2.15	45.94	51.9	45.7	46.18	51.65	52.15
360	0.57	1.78	47.1	55.58	46.89	47.31	55.37	55.79
Transit-stub Graph								
Destination Nodes	Standard Deviation		Mean Multicast Time		99% Confidence Interval			
	SSH	Brute-force	SSH	Brute-force	SSH		Brute-force	
40	1.22	3.55	14.68	27.1	14.24	15.12	26.69	27.51
80	0.94	4.25	15	28.34	14.66	15.34	27.85	28.83
140	0.85	3.45	16.7	28.32	16.39	17.01	27.92	28.72
200	0.86	6.27	18.24	32.32	17.93	18.55	31.6	33.04
280	0.86	5.16	16.88	28.98	16.57	17.19	28.39	29.57
360	0.28	3.23	18.96	29.96	18.86	19.06	29.59	30.33

Table 5: 99% Confidence Intervals for Time in Graph Models 1

Pure Random Graph								
Destination Nodes	Standard Deviation		Mean Multicast Traffic		99% Confidence Interval			
	SSH	Brute-force	SSH	Brute-force	SSH		Brute-force	
40	0.5	0.64	5.5	26.59	5.32	5.68	26.52	26.66
80	0.7	0.72	8.77	26.72	8.51	9.03	26.64	26.8
140	0.67	0.77	13.25	26.77	13.01	13.49	26.68	26.86
200	0.66	0.82	16.57	26.82	16.33	16.81	26.73	26.91
280	0.69	0.83	20.95	26.83	20.7	21.2	26.73	26.93
360	0.67	0.85	25.04	26.85	24.8	25.28	26.75	26.95
Hierarchical Graph								
Destination Nodes	Standard Deviation		Mean Multicast Traffic		99% Confidence Interval			
	SSH	Brute-force	SSH	Brute-force	SSH		Brute-force	
40	2.17	2.21	35.52	89	34.73	36.31	88.75	89.25
80	2.43	0.99	47.72	89.85	46.83	48.61	89.74	89.96
140	2.05	0.67	60.49	90.3	59.74	61.24	90.22	90.38
200	2.02	0.6	70.31	90.54	69.57	71.05	90.47	90.61
280	1.14	0.65	79.88	90.65	79.46	80.3	90.58	90.72
360	0.73	0.67	87.42	90.67	87.15	87.69	90.59	90.75
Transit-stub Graph								
Destination Nodes	Standard Deviation		Mean Multicast Traffic		99% Confidence Interval			
	SSH	Brute-force	SSH	Brute-force	SSH		Brute-force	
40	1.72	1.94	21.5	84.47	20.87	22.13	84.25	84.69
80	2.14	1	36.14	85.48	35.36	36.92	85.36	85.6
140	1.93	0.76	51.35	85.9	50.65	52.05	85.81	85.99
200	1.66	0.67	61.72	86.02	61.12	62.32	85.94	86.1
280	1.2	0.52	73.45	86.13	73.01	73.89	86.07	86.19
360	1	0.34	82.4	86.16	82.04	82.76	86.12	86.2

Table 6: 99% Confidence Intervals for Traffic in Graph Models 1

Tables 9 and 10 represent the 99% confidence intervals over all the different simulations performed for multicast time and traffic over various graphs using multiple sets of source and destination nodes. The sample size in this case is 7500 for SSH algorithm and 75000 (7500 x 10) for Brute-force algorithm.

Pure Random Graph								
Destination Nodes	Standard Deviation		Mean Multicast Time		99% Confidence Interval			
	SSH	Brute-force	SSH	Brute-force	SSH		Brute-force	
40	1.32	1.77	14.98	13.9	14.5	15.46	13.7	14.1
80	1.53	2.14	16.06	14.48	15.5	16.62	14.23	14.73
140	1.2	1.87	16.26	14.78	15.82	16.7	14.56	15
200	0.65	1.18	17.06	14.7	16.82	17.3	14.56	14.84
280	0.5	1.84	18.7	15.64	18.52	18.88	15.43	15.85
360	0.28	1.28	14.96	14.66	14.86	15.06	14.51	14.81
Hierarchical Graph								
Destination Nodes	Standard Deviation		Mean Multicast Time		99% Confidence Interval			
	SSH	Brute-force	SSH	Brute-force	SSH		Brute-force	
40	5.53	6.6	47.44	59.8	45.43	49.45	59.04	60.56
80	1.18	4.26	53.62	70.56	53.19	54.05	70.07	71.05
140	1.17	4.3	53.94	65.32	53.51	54.37	64.82	65.82
200	1.2	2.48	48.36	56.58	47.92	48.8	56.29	56.87
280	1.1	2.17	49.48	54.48	49.08	49.88	54.23	54.73
360	0.42	3.27	60.94	66.54	60.79	61.09	66.16	66.92
Transit-stub Graph								
Destination Nodes	Standard Deviation		Mean Multicast Time		99% Confidence Interval			
	SSH	Brute-force	SSH	Brute-force	SSH		Brute-force	
40	0.9	2.03	13.48	19.76	13.15	13.81	19.53	19.99
80	0.97	2.49	17.18	25.78	16.83	17.53	25.49	26.07
140	0.69	2.85	17.72	26.34	17.47	17.97	26.01	26.67
200	0.78	4.51	18.48	29.36	18.2	18.76	28.84	29.88
280	0.59	3.22	17.26	25.82	17.05	17.47	25.45	26.19
360	0.9	1.66	21.44	23.24	21.11	21.77	23.05	23.43

Table 7: 99% Confidence Intervals for Time in Graph Models 2

Pure Random Graph (Sample Size N = 500 and 50)								
Destination Nodes	Standard Deviation		Mean Multicast Traffic		99% Confidence Interval			
	SSH	Brute-force	SSH	Brute-force	SSH		Brute-force	
40	0.72	0.61	4.98	25.17	4.72	5.24	25.1	25.24
80	0.73	0.47	8.05	25.25	7.78	8.32	25.2	25.3
140	0.58	0.36	12.25	25.33	12.04	12.46	25.29	25.37
200	0.69	0.38	15.74	25.37	15.49	15.99	25.33	25.41
280	0.74	0.39	19.84	25.39	19.57	20.11	25.35	25.43
360	0.62	0.39	23.63	25.39	23.4	23.86	25.35	25.43
Hierarchical Graph (Sample Size N = 500 and 50)								
Destination Nodes	Standard Deviation		Mean Multicast Traffic		99% Confidence Interval			
	SSH	Brute-force	SSH	Brute-force	SSH		Brute-force	
40	2.18	2.69	29.76	80.44	28.97	30.55	80.13	80.75
80	2.18	1.01	41.58	81.81	40.79	42.37	81.69	81.93
140	1.61	0.61	53	82.17	52.41	53.59	82.1	82.24
200	1.56	0.48	62.48	82.32	61.91	63.05	82.26	82.38
280	1.1	0.4	71.5	82.4	71.1	71.9	82.35	82.45
360	0.91	0.43	79.3	82.43	78.97	79.63	82.38	82.48
Transit-stub Graph (Sample Size N = 500 and 50)								
Destination Nodes	Standard Deviation		Mean Multicast Traffic		99% Confidence Interval			
	SSH	Brute-force	SSH	Brute-force	SSH		Brute-force	
40	1.62	1.4	21.21	79.05	20.62	21.8	78.89	79.21
80	1.97	0.91	34.69	79.6	33.97	35.41	79.5	79.7
140	1.7	0.72	48.02	79.9	47.4	48.64	79.82	79.98
200	1.77	0.61	57.79	80	57.15	58.43	79.93	80.07
280	1.19	0.46	68.16	80.08	67.73	68.59	80.03	80.13
360	0.89	0.25	76.45	80.11	76.13	76.77	80.08	80.14

Table 8: 99% Confidence Intervals for Traffic in Graph Models 2

Pure Random Graph								
Destination Nodes	Standard Deviation		Mean Multicast Time		99% Confidence Interval			
	SSH	Brute-force	SSH	Brute-force	SSH		Brute-force	
40	1.69	1.77	13.73	14.38	13.68	13.78	14.36	14.4
80	1.49	1.63	14.58	15.2	14.54	14.62	15.18	15.22
140	1.6	1.65	15.32	15.86	15.27	15.37	15.84	15.88
200	1.58	1.62	15.46	16.23	15.41	15.51	16.21	16.25
280	1.72	1.68	15.88	16.5	15.83	15.93	16.48	16.52
360	1.67	1.48	15.96	16.59	15.91	16.01	16.58	16.6
Hierarchical Graph								
Destination Nodes	Standard Deviation		Mean Multicast Time		99% Confidence Interval			
	SSH	Brute-force	SSH	Brute-force	SSH		Brute-force	
40	10.99	15.38	43.15	60.47	42.82	43.48	60.33	60.61
80	10.37	14.76	43.6	60.02	43.29	43.91	59.88	60.16
140	9.21	13.24	44.4	60.4	44.13	44.67	60.28	60.52
200	11.27	15.75	46.29	62.79	45.95	46.63	62.64	62.94
280	10.46	14.45	46.27	61.27	45.96	46.58	61.13	61.41
360	10.41	14.4	46.51	61.26	46.2	46.82	61.12	61.4
Transit-stub Graph								
Destination Nodes	Standard Deviation		Mean Multicast Time		99% Confidence Interval			
	SSH	Brute-force	SSH	Brute-force	SSH		Brute-force	
40	2.12	5.02	15.32	30.22	15.24	15.4	30.16	30.28
80	2.32	4.77	17.61	31.59	17.53	17.69	31.54	31.64
140	2.21	4.88	18.57	32.13	18.49	18.65	32.07	32.19
200	2.14	4.96	19.37	32.6	19.29	19.45	32.54	32.66
280	2.18	4.52	20.27	32.97	20.19	20.35	32.92	33.02
360	2.55	4.76	20.86	32.71	20.77	20.95	32.66	32.76

Table 9: 99% Confidence Intervals of Overall Mean Multicast Time.

Pure Random Graph								
Destination Nodes	Standard Deviation		Mean Multicast Traffic		99% Confidence Interval			
	SSH	Brute-force	SSH	Brute-force	SSH		Brute-force	
40	0.74	1.43	5.19	26.01	5.17	5.21	26	26.02
80	0.85	1.42	8.52	26.15	8.49	8.55	26.14	26.16
140	0.99	1.43	12.64	26.22	12.61	12.67	26.21	26.23
200	1.12	1.45	16.21	26.25	16.18	16.24	26.24	26.26
280	1.28	1.46	20.52	26.27	20.48	20.56	26.26	26.28
360	1.45	1.46	24.45	26.28	24.41	24.49	26.27	26.29
Hierarchical Graph								
Destination Nodes	Standard Deviation		Mean Multicast Traffic		99% Confidence Interval			
	SSH	Brute-force	SSH	Brute-force	SSH		Brute-force	
40	2.89	3.54	31.23	82.39	31.14	31.32	82.36	82.42
80	3.12	3.18	42.88	83.33	42.79	42.97	83.3	83.36
140	3.13	3.13	54.79	83.68	54.7	54.88	83.65	83.71
200	3.27	3.11	63.62	83.82	63.52	63.72	83.79	83.85
280	3.33	3.11	72.93	83.91	72.83	73.03	83.88	83.94
360	3.2	3.11	80.63	83.95	80.53	80.73	83.92	83.98
Transit-stub Graph								
Destination Nodes	Standard Deviation		Mean Multicast Traffic		99% Confidence Interval			
	SSH	Brute-force	SSH	Brute-force	SSH		Brute-force	
40	1.84	3.89	22.23	82.58	22.16	22.3	82.54	82.62
80	2.35	3.66	35.99	83.44	35.9	36.08	83.4	83.48
140	2.88	3.62	50.28	83.78	50.18	50.38	83.74	83.82
200	3.21	3.62	60.65	83.91	60.53	60.77	83.87	83.95
280	3.53	3.61	71.44	83.99	71.31	71.57	83.95	84.03
360	3.59	3.58	80.23	84.03	80.1	80.36	83.99	84.07

Table 10: 99% Confidence Intervals of Overall Mean Multicast Traffic.

Chapter 5

Conclusion and Future Work

This thesis presents a multicasting algorithm namely SSH for any given network. The algorithm works by first constructing a multicast tree by using heuristics and then multicasting based on these heuristic values. For comparison purposes, the thesis also defines a simple store-and-forward algorithm named Brute-force, which basically multicasts randomly to any of its child nodes.

These algorithms have been compared in three different network topologies that simulate the real world—Pure random model, Hierarchical model and the Transit-stub model. The SSH algorithm on an average performs better in both the multicast time and multicast traffic parameters. However, in case of hierarchical graph model, some instances of the graph have better performance in Brute-force algorithm compared to the SSH algorithm. The biggest advantage of the SSH algorithm is in using the network resources efficiently and reducing the traffic generated for multicasting. The algorithm

also has better mean multicasting time in case of hierarchical graphs and transit-stub graphs.

Since the multicast problem is NP-hard and a single multicast algorithm cannot function optimally in every case, it is desirable to use heuristics that are based on good strategies, which obtain near optimal results in one or the other practical aspects of the network topology under certain circumstances. More attention and research need to be directed to this field to develop good multicasting algorithms under specific circumstances such as different architecture, applications and network technologies. The following is some of the work need to be done based on our research in future.

- Extend and/or implement these algorithms for higher dimensional network models and then analyze the multicast performance based on the changing trend of the size and dimensions of the network.
- Adapt the multicast algorithm with specific multicomputer hardware architecture, deadlock preventing, fault tolerant and traffic load balancing algorithms that can then be used in real world applications.
- Optimize the existing algorithm or develop new algorithms for Multimessage multicast environment by using a better technique to generate the multicast tree. The structured network models can have better performance when used with a naming convention that can easily identify them from a group of nodes. This can be used in conjunction with other similar techniques to create a better algorithm.
- Develop pro-traffic multicast algorithms, i.e., algorithms that target to minimize traffic, since the algorithms developed in this thesis are all pro-time.

- Develop multicast algorithms that specially are optimized for hierarchical network models that simulate the Internet and the Intranet.

Bibliography

[1] Hovhannes A. Harutyunyan and Bin Shao. A Heuristic for K-Broadcasting in Arbitrary Networks. In *Proceedings of the Seventh International Conference on Information Visualization*, 2003 IEEE.

[2] Mathew Doar and Ian Leslie. How Bad is Naive Multicast Routing? In *Proceedings of IEEE INFOCOM '93, Twelfth Annual Joint Conference of the IEEE Computer and Communications Societies*. pages 82–89, 1993.

[3] Bernard M. Waxman. Routing of Multipoint Connections. *IEEE Journal on Selected Areas in Communications*, 6(9):1617–1622, 1988.

[4] Liming Wei and Deborah Estrin. The Trade-offs of Multicast trees and Algorithms. In *International Conference on Computer Communications and Networks*, August 1994.

[5] F.F. Rivera, O. Plata, E.L. Zapata. Broadcasting algorithm in Computer Networks: Accumulative depth. *IEEE Proceedings*—Volume 137, Issue 6, Nov 1990.

[6] Hovhannes A. Harutyunyan and Bin Shao. Efficient Heuristics for Message Dissemination in Networks. In *Proceedings of the 25th IASTED International Multi-Conference,* Feb 2007.

[7] Xiaola Lin and Lionel M. Ni. Multicast Communication in Multicomputer Networks, *IEEE Transactions of Parallel and Distributed Systems*, vol. 4, pp. 1105–1117, October 1993.

[8] Lionel M. Ni and Philip K. McKinley. A Survey of Wormhole Routing Techniques in Direct networks. *IEEE Computer* 26(2), pp. 62–76, 1993.

[9] David F. Robinson, Philip K. McKinley, Betty H. C. Cheng, Optimal Multicast Communication in Wormhole-Routed Torus Networks, *IEEE Transactions on Parallel and Distributed Systems*, Volume 6, Issue 10, pp. 1029–1042, October 1995.

[10] Cory J. Hoelting, Dale A. Schoenefeld and Roger L. Wainwright. A Genetic Algorithm for the Minimum Broadcast time problem using a Global Precedence Vector. *Proceedings of the 1996 ACM symposium on Applied Computing.*

[11] P. Scheuermann, G. Wu. Heuristic Algorithm for Broadcasting in Point-to-Point Computer Networks, *IEEE Transactions on Computers*, Vol. 33, no. 9, pp. 804–811, Sept 1984.

[12] Youran Lan, Abdol-Hossein Esfahanian and Lionel M. Ni. Multicast in Hypercube Multiprocessors. *In Proceedings of 1988 IEEE Seventh Annual International Phoenix conference on Computers and Communications.*

[13] Y. Choi, A. Esfahanian and L. Ni. One-to-K Communication in Distributed memory multiprocessors. In *Proceedings of the 25th Annual Allerton Conference on Communication, Control and Computing*, September 1987.

[14] P. Fraigniaud and E. Lazard. Methods and Problems of Communication in Usual Networks. *Discrete Applied Mathematics* 53 (1994), 79–133.

[15] F. Harary, Graph Theory, *Addison-Wesley*, Reading, MA, 1972.

[16] Prasant Mohapatra. Wormhole Routing Techniques for Directly Connected Multicomputer Systems, *ACM Computing Surveys (CSUR)*, Volume 30, Issue 3, pp. 374–410, 1998.

[17] E.W. Zegura, K. Calvert and S. Bhattacharjee. How to Model an Internetwork. *IEEE INFOCOM*, San Francisco, CA, 1996.

[18] K. Bharath Kumar and J. M Jaffe. Routing to Multiple destinations in Computer networks. *IEEE Transactions*, 1983, COM-31, (3), pp. 343–351.

[19] K. Jakobs and U. Quernheim. Multicast Communication in Networks with Arbitrary Topology. *Technical University of Aachen, Federal Republic of Germany.*

[20] P.K. McKinley, et al. Unicast-Based Multicast Communication in Wormhole Routed Networks. *Proceedings of 1992 International Conference of Parallel Processing*, Vol II, IEEE CS Press, Los Alamitos, California, Order No. 3155, 1992, pp. 10–19.

[21] W.C. Athas and C.L. Seitz. Multi-computers: Message Passing Concurrent Computers, *Innovative Technology for Computing Professionals, Computer*. Vol. 21, No. 8, Aug. 1988, pp. 9–25.

[22] G.T. Byrd, N.P. Saraiya and B.A. Delagi. Multicast Communication in Multiprocessor Systems. *In Proceedings of 1989 International Conference of Parallel Processing*, pp. I-196–I-200.

[23] C.L. Seitz, J. Seizovic and W.K. Su. The C Programmer's Abbreviated Guide to Multicomputer Programming. *Department of Computer Science, California Institute of Technology, Tech. Rep. Caltech-CS-TR-88-1*, Jan 1988.

[24] National Institute of Standards and Technology, *"Dictionary of Algorithms and Data Structures"*, http://www.nist.gov/dads.

[25] Free Software Foundation Inc, *"Wikipedia, the free encyclopedia"*, http://en.wikipedia.org/wiki/Main_Page.

[26] The Network Simulator – ns-2 website at http://www.isi.edu/nsnam/ns/

[27] J. Y. Lee Park, H. A. Choi, N. Nupairoj and L. M. Ni. Construction of optimal multicast trees based on the parameterized communication model. *In International Conference on Parallel Processing*, Vol.1, pages180–187, 1996.f

[28] A. Yassin Al-Dubai, M. Ould-Khaoua, L. M. Mackenzie. An Efficient Path-Based Multicast Algorithm for Mesh Networks. *In International Parallel and Distributed Processing Symposium (IPDPS)*, p. 283, 2003.

[29] Hovhannes A. Harutyunyan and X. Liu. New Multicast Algorithms in Mesh-connected Networks, *International Symposium on performance Evaluation of computer and Telecommunication Systems (SPECTS)*, Montreal, Canada, pp. 284–291, 2003

[30] Honge Wang and Douglas M. Blough. Multicast in Wormhole-Switched Torus Networks Using Edge-Disjoint Spanning Trees. *Journal of Parallel and Distributed Computing*, Volume 61, Issue 9, September 2001, Pages 1278–1306

[31] Shih-Hsien Sheu, Chang-Biau Yang. Multicast Algorithms for Hypercube Multiprocessors. *Journal of Parallel and Distributed Computing,* Volume 61, Issue 1, pp. 137–149, 2001

[32] Norman E. Fenton and Shari Lawrence Pfleeger. Software Metrics – A Rigorous & Practical Approach. Second Edition. *International Thomson Publishing*, pp. 118–122, 1997.

[33] William Mendenhall and Terry Sincich. Statistics for Engineering and the Sciences. Fourth Edition. *Prentice Hall Inc*. Pages 352–367.