UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

PROTOCOLS FOR COLLABORATIVE APPLICATIONS ON OVERLAY NETWORKS

A DISSERTATION

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for the

degree of

Doctor of Philosophy

By

SHANKAR MADHAB BANIK
Norman, Oklahoma
2006

Downloaded from SHAREOK. Persistent Handle URL: https://hdl.handle.net/11244/989

UMI Number: 3207081



UMI Microform 3207081

Copyright 2006 by ProQuest Information and Learning Company.
All rights reserved. This microform edition is protected against
unauthorized copying under Title 17, United States Code.

ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

PROTOCOLS FOR COLLABORATIVE APPLICATIONS ON OVERLAY NETWORKS

A DISSERTATION APPROVED FOR THE
DEPARTMENT OF COMPUTER SCIENCE

BY

_____

DR. SRIDHAR RADHAKRISHNAN (Chair)

_____

DR. S. LAKSHMIVARAHAN

_____

DR. SUDARSHAN K. DHALL

_____

DR. K. THULASIRAMAN

_____

DR. MARILYN BREEN

© Copyright by SHANKAR MADHAB BANIK 2006
All Rights Reserved.

Acknowledgements

I wish to express my sincere gratitude to my dissertation advisor Dr. Sridhar Radhakrishnan for his continuous guidance and inspiration without which this work would have been impossible. The mutual discussions and his ideas have been the building blocks of this work. I am extremely thankful for his patience, tolerance and encouragement throughout the course of this dissertation.

Next I would like to thank Dr. S. Lakshmivarahan, Dr. Sudarshan K. Dhall, Dr. K. Thulasiraman and Dr. Marilyn Breen for serving in my dissertation committee. Also I would like to thank Dr. Chandra N. Sekharan of Loyola University at Chicago, exchanging research ideas and discussing problems with him have enriched my knowledge and given me important ideas for my research. I would like to thank Dr. John K. Antonio, Director of School of Computer Science, The University of Oklahoma, for his continuous support throughout my graduate studies.

I am grateful to friends in Norman for their support and well wishes. Especially I would like to thank Md. Javed, Jonyung Kim, Tao Zheng, Aravind Mohanoor and Brian Veale.

I would like to thank all the members of my family especially my parents Biswa Madhab and Jyotsna Banik who always trusted me and believed in my abilities.

Last but not the least I would like to thank my wife, Antara. She stood by me through thick and thin while I worked towards completion of this dissertation.

Contents

1. Introduction

1.1. Overlay Networks ……1

1.2. Collaborative Applications ……....4

1.3. Floor Control in Collaborative Applications ……4

1.4. Multicasting for Data Delivery ……8

1.5. Limitations of Traditional Multicasting Models ……12

1.6. Embedding Multicasting Trees on the Overlay Network ……14

1.7. Organization of the Dissertation ……17

2. Floor Control in Collaborative Applications

2.1. Introduction ……18

2.2. Overview of the Solution …….....19

2.3. Optimal Participating Chain for Contention Phase ……23

2.4. Protocols for Resolving Floor Contention ……..32

2.5. Data Delivery Phase …….....44

2.6. Analytical Evaluation ……..47

2.7. Simulation Experiments ……..52

2.8. Related Work ……..54

2.9. Summary ……57

3. Multicasting with Delay Variation

3.1. Introduction ……59

3.2. Formal Definition of the Problem ……...61

3.3. Literature Review ……63

3.4. Motivating Example ……67

3.5. Our Heuristic: Chains ……..71

3.6. Performance Evaluation ……..80

3.7. Dynamic Reorganization of Multicasting Sub-network …… 87

3.8. Summary ……89

4. Duplicating Delays in Overlay Multicasting

4.1. Introduction ……91

4.2. Formal Definition of the Problem ……..96

4.3. Algorithm and its Time Complexity ……113

4.4. Lower Bound Results ……114

4.5. Simulation Experiments ……115

4.6. Summary ……...118

5. Embedding Multicasting Trees on the Overlay Network

5.1. Introduction ……..119

5.2. Formal Definition of the Problem ……123

5.3. Our Approach ……124

5.4. Algorithm and its Time Complexity …… ……127

5.5. Summary ……...131

6. Conclusion and Future Work

6.1. Summary of Contribution ……132

6.2. Scope for Future Research ……135

7. **Bibliography** ……137

List of Figures

1.1 Peer-to-peer overlay network ……3

1.2 Proxy based overlay network ……4

1.3 An example of a tree network specified by an application designer ……...15

2.1 A graph G = (V, E) ……..25

2.2 A tree with node A as root ……..29

2.3 The tree after bottom-up computation of $l(x)$ for each non-leaf node x ……..29

2.4 The rearranged tree with the in-order traversal ……...30

2.5 State Diagram for ALOHA on a participating chain ……...37

2.6 A participating chain with request_flow path and signal_flow path ……...38

2.7 State Diagram for DQDB on a participating chain …….43

2.8 A participating chain where an NSN appears more than once ……43

2.9 Timing diagram for floor control signal by NSNs A and B in ALOHA …….46

2.10 Buffering end-user data at the root for DQDB ……..47

2.11 Comparing Efficiencies of Different Floor Control Protocols with Analytical Model ……52

2.12 Efficiencies VS System Request Rate for Different Floor Control Protocols from

Simulation Experiments ……54

3.1 A graph with S as source node and D_1, D_2, D_3 as destination nodes ……..61

3.2 Example of a network with link delays. Nodes in double circles are the destination nodes and node V_s is the source node ……..68

3.3 The multicasting sub-network with a delay variation of 3 and a delay bound 50 obtained after merging the paths (d), (g), and (j) in Table 3.1……..71

3.4 Delay variation graph and execution time graph for 5% of the nodes as destinations……. ……..84

3.5 Delay variation graph and execution time graph for 10% of the nodes as destinations……. ……..84

3.6 Delay variation graph and execution time graph for 15% of the nodes as destinations……. ……..84

3.7 Comparison of Execution Times of DPDVB and Chains for different values of Δ (End-to-End Delay bound) with number of nodes 100 ……86

3.8 Execution time for finding k-shortest paths when the number of nodes is 50 and 100 ……86

3.9 k vs End-to-End Delay bound for networks with average node degree ……86

4.1 Multicasting tree with link delays and duplicating delay vectors at each node ….94

4.2 Hook-up fan ……..102

4.3 Hook-up fan H with duplicating delay vectors ……103

4.4 Fan $F(j)$ with new duplicating delay vectors ……104

4.5 Fan $F(j)$ with feasible duplicating delay vectors ……...105

4.6 Hook-up fan H with optimal multicasting tree assignments ……106

4.7 Hook-up fan *H* with optimal multicasting ……109

4.8 Multicast tree with duplicating delay vectors at nodes and assuming that the link delays are the same on all links ……...110

4.9 Optimal Solutions for fans b, c and d ……111

4.10 The Hook-up fans and its Optimal Solution ……112

4.11 The optimal multicasting tree ……..113

4.12 Experimental Setup for Multicasting Application ……..116

4.13 Plotting of Different Orders of Multicasting VS Delay ……..117

5.1 A shortest path tree *I* and a multicasting tree *T* …….....122

5.2 An example of $H_r(v)$ and $T_{r'}(u)$ …….126

List of Tables

2.1 Parameters used in the analysis ……..48

2.2 Parameters used in the simulation ……...53

3.1 List of paths from V_S to V_2, V_5, V_8 and their corresponding end-to-end delays …69

3.2 Execution Time vs Number of Nodes with 10% of the nodes in the Multicast Group…….85

3.3 Execution time vs % of nodes in the Multicast Group (Total Nodes = 80) ……85

3.4 Comparison of Execution Times of DPDVB and Chains with 10% of the nodes in the Multicast Group……85

4.1 Table demonstrating that order matters in multicasting ……117

Abstract

Computer supported collaborative applications are gaining popularity among Internet users who are geographically dispersed. Examples of this kind of application range from video conferencing, video-on-demand, distributed database replication, distributed interactive simulations, online multiplayer games, and peer-to-peer file sharing systems. These types of distributed applications call for efficient group communication which entails determining routes that are independent of the underlying network. To meet the demands of these distributed applications, there have been increased research efforts in the development of network protocols that can be executed at the application layer. These protocols are built for virtual networks named as *overlay networks*. In an overlay network, the nodes are the hosts that participate in the distributed application and the links are paths in the Internet that consist of several routers along the path. Overlay networks provide a flexible and deployable approach for many distributed applications.

Our research is focused on the development of algorithms for the construction of overlay networks that meet the demands of the distributed applications. In addition, we have provided network protocols that can be executed on these overlay networks for a chosen set of collaborative applications: *floor control* and *multicasting*. Our contribution in this research is four fold.

First, we consider the *floor control problem* wherein the participating users coordinate among themselves to gain exclusive access to the communication channel. To solve the floor control problem, we present an implementation and evaluation of distributed Medium Access Control (MAC) protocols on overlay networks. As an initial step in the implementation of these MAC protocols, we propose an algorithm to construct an efficient communication channel among the participating users in the overlay network. We also show that our implementation scheme (one of the first among decentralized floor control protocols) preserves the *causal ordering* of messages.

Second, we address the problem of designing multicasting sub-network for collaborative applications using which messages are required to arrive at the destinations within a specified delay bound and all the destinations must receive the message from a source at 'approximately' the same time. The problem of finding a multicasting sub-network with delay and delay-variation bound has been proved to be NP-Complete in the literature and several heuristics have been proposed. We have designed and implemented a fast heuristic and our extensive empirical studies indicate that our heuristic uses significantly less run-time in comparison with the best-known heuristics while achieving the tightest delay variation for a given end-to-end delay bound.

Third, we address the limitations of traditional multicasting models. Towards this, we propose a model where a source node has different switching time for each child node and the message arrival time at each child depends on the order in which the source chooses to send the messages. This model captures the heterogeneous nature of

communication links and node hardware on the overlay network. Given a multicast tree with link delays and generalized switching delay vectors at each non-leaf node, we provide an algorithm which schedules the message delivery at each non-leaf node in order to minimize the delay of the multicast tree. Our algorithm uses the concept of min-max matching problem on bipartite graphs. We also show an important lower bound result that states that optimal multicast switching delay problem is as hard as min-max matching problem on bipartite graphs.

Fourth, we address the problem of finding an arbitrary application designer specific overlay network on the Internet. This problem is equivalent to the problem of subgraph homeomorphism and it is NP-Complete. We have designed a polynomial-time algorithm to determine if a delay constrained multicasting tree (call it a *guest*) can be homeomorphically embedded in a general network (call it a *host*). A delay constrained multicasting tree is a tree wherein the link weights correspond to the maximum allowable delay between the end nodes of the link and in addition, the link of the guest should be mapped to a shortest path in the host. Such embeddings will allow distributed application to be executed in such a way that application specific quality-of-service demands can be met.

Chapter 1

Introduction

1.1 Overlay Networks

Internet has made a tremendous impact on human civilization by providing global connectivity to vast number of end users who are geographically dispersed from each other. Researchers believe that a large portion of Internet's success was due to its simple underlying network protocol (IP). But success of Internet also brings with it a limitation that it is very difficult to deploy a new protocol or add a new service at the network layer. Adding a new service at the network layer implies changing all the routers in the Internet around the world which is not feasible. Applications like video conferencing, large-scale distributed interactive simulation, online games, distributed database replication would like to perform efficient group communication by choosing a route which is not dictated by the underlying network. So the efficient

implementation of these applications requires the assistance of the routers on the underlying network. This may include anywhere from network layer modification to requiring the routers to perform additional special functions based on packet header information. These requirements impose limitations in terms of scalability, network management, and deployment and hence have been recognized as major impediments for the wide acceptance by the Internet community.

To alleviate this problem, recent research trend is to develop and implement network layer protocols at the application layer. This method will give the flexibility to the users to develop network protocols that suit their application needs. Clearly, application programs can reside only at the end-hosts. Taking a note of this, network applications build virtual networks named *overlay networks* [1-6]. An overlay network is a virtual network deployed over an existing network. In an overlay network, each individual link which connects two nodes can comprise of several routers and hosts in the underlying physical network. Overlay networks provide a flexible and deployable approach for applications. When a better routing or a control protocol is demanded by an application, it can be easily deployed in the application level without changing the lower level Internet protocols.

In an overlay network the end-users self-organize into a network architecture and share the responsibility for creating, consuming, and forwarding messages to other end-users. Each edge in this network corresponds to a unicast path between two end-hosts in the underlying network. Two types of architectures have been proposed by the

researchers for the overlay network: peer-to-peer architecture and proxy based architecture. In the peer-to-peer architecture (Figure 1.1), all end-hosts are connected to each other. The network functionality of the overlay is pushed to end-hosts. Napster [7] and Gnutella [8] are the examples of peer to peer architecture. The disadvantage of this architecture is that it puts additional burden on end-hosts as each end-host has to maintain the routing and group management information at its end. In the proxy based architecture (Figure 1.2), networking service is provided through a set of distributed nodes called Network Service Nodes (NSN) (Multicasting on overlay networks [1, 2] refer to these nodes as Multicast Service Nodes (MSN)). The NSNs communicate with end-hosts and with each other using standard unicast mechanism. M-Bone [9] and X-Bone [10] are examples of proxy-based overlay networks. Most of the research work dealing with multicasting on overlay networks [1, 2, 4] uses proxy based architecture.

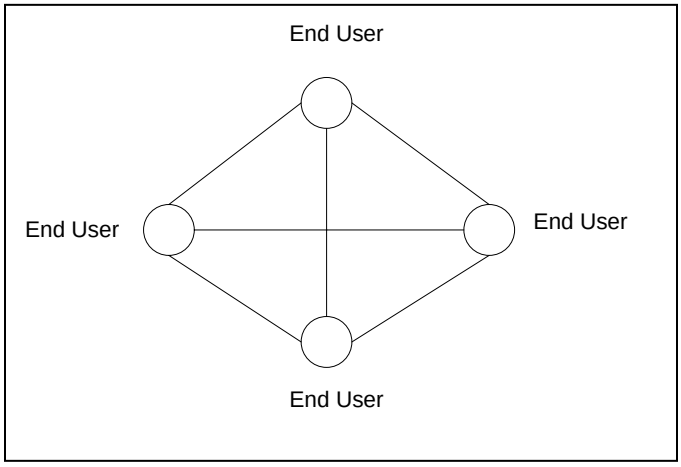

Figure 1.1: Peer-to-peer overlay network

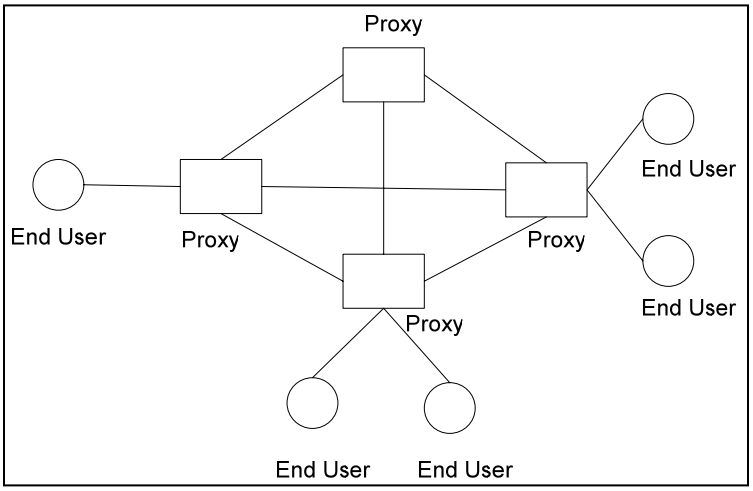

Figure 1.2: Proxy-based overlay network

1.2 Collaborative Applications

Collaborative Applications are a class of applications that involve sharing of data between a set of geographically distributed users connected to each other through a network. Examples of such applications include video conferencing, collaborative design and simulation, distributed database replication, and online games. In this type of applications, users coordinate their activities so as to achieve a common goal.

1.3 Floor Control in Collaborative Applications

In Collaborative Applications, the users typically share some resources and it is required that at any given instance, only one user has exclusive access to a given

common resource. One such shared resource is the underlying network that the distributed users employ to communicate with one another. The problem of providing a user the exclusive access to the communication channel facilitated by the underlying network is termed as the *floor control* problem [11, 12] and the channel is referred to as the *floor*. Usually, the user who wins the floor will use an efficient mechanism (e.g. multicasting) to send a sequence of messages to all the other users in the collaborative application.

A simple mechanism to implement floor control on an overlay network would be to choose a centralized NSN (the controller NSN) in the overlay network that will control the access to the network. Each end-host that has a message to send, first sends a request for permission to access the channel to its NSN which in turn forwards the request to the controller NSN that schedules the access request. This centralized approach albeit simple runs into several problems: the links to the controller NSN can be congested with requests thereby slowing the processing of requests and failure of the controller NSN results in the failure of the entire application. Dommel and Garcia [11] presented a centralized approach and a novel taxonomy to analyze and compare existing floor control protocols for the regular internet. Towards this, the authors have proposed the first tree based group coordination protocol where the logical control tree is correlated to the underlying multicasting tree. When a node wants to acquire the floor, it sends request to the root of the control tree which is the current floor handler. When the root wants to relinquish the floor, it sends a floor grant signal to the requester. The root also informs other nodes in the tree about the new floor handler.

The control tree is adjusted to make the new floor handler as the root of the tree. The authors have claimed that the proposed tree based protocol is dynamically centralized in the sense that the controller node is dynamically selected instead of using a single central node for the entire session. The disadvantage of this protocol is that the controller node will always be biased towards the nodes which are nearer to the root of the tree.

To overcome the deficiencies of the centralized approach, decentralized approaches as presented in the MAC protocols for LANs can be adapted with appropriate implementation considerations. In this research, we consider two well known MAC protocols: ALOHA and Distributed Queue Dual Bus (DQDB) and show how these protocols can be efficiently implemented on an overlay network. We consider ALOHA for floor control as it was the first randomized solution to channel allocation problem in MAC layer and is the basis of IEEE 802.3 standard protocol. The DQDB protocol is chosen for floor control as it is an elegant solution for a distributed network which achieves First In First Out (FIFO) order without having a centralized queue. Certain group communication applications such as video conferencing, collaborative design and simulation and online games require FIFO ordering of requests to maintain fairness among the users.

The first step in the implementation of these MAC protocols on an overlay network is to construct an efficient communication channel among the NSNs in the overlay network. The efficiency of the communication channel is directly proportional

to the maximum of the end-to-end delay between all pairs of NSNs. Other metrics of efficiency such as average waiting time at the intermediate NSNs and loss probability can also be used to construct this communication channel. The communication channel can be thought of as a token-bus network where each node is connected through a bus. Since in an overlay network, each end-host is attached to a NSN and NSNs are connected to each other, our goal is to construct an efficient spanning sub-network of NSNs for coordinating floors among the end-hosts. We assume that each end hosts is connected to its nearest NSN using the shortest unicast path in the underlying physical network.

In this research, we present efficient implementation of distributed protocols for the floor control problem on a proxy based overlay network. The algorithm previously proposed for the floor control problem on wide area networks [11-15] can be extended to the case of overlay networks but they do not take into account the end-to-end delays of the underlying communication infrastructure which are crucial on an overlay network. They also do not consider the causal ordering of messages which guarantees that messages are received in the sending order. We analyze the efficiency of our implementation of distributed floor control protocols and show that our implementation preserves causal ordering of messages. As an initial step in the implementation of floor control protocols, we propose an algorithm to construct an efficient communication channel among the NSNs.

1.4 Multicasting for Data Delivery

In a collaborative application, when an end-host acquires the floor, it wants to deliver data to other participating end-hosts. We propose to use *multicasting* for data delivery as multicasting is an efficient communication mechanism in which a source host sends the same message to a group of destination hosts, called the *multicasting* group. The general strategy of accomplishing this task is to construct a rooted tree *T* called the *multicast tree* that contains the source as the root and the destination hosts as the leaf nodes. The primary advantage of using the multicast is that it conserves network bandwidth. Contrasted with the *unicast* mechanism where separate messages are sent to each destination host from the source host, multicasting avoids sending the same message multiply over links that are common to a source and different destinations.

Collaborative applications such as video conferencing, online games, interactive simulations, distributed database replications require that messages should arrive at the destinations within a specified delay bound. Furthermore, these applications also require that the destinations receive the message from the source at approximately the same time. In applications for teleconference, the data sent from speaker's end should reach all the participants at the same time, which means the delay variation among the paths from the speaker to the participants should be minimum. Certain critical database applications (e.g. hospital information systems [16], stock brokerage database) require that data should be replicated at all sites at the same time,

otherwise data will be temporally inconsistent. For example, consider a telemedicine system with distributed databases which stores the medical test results of patients. Physicians with varied expertise from different places make decisions on patients' medication and procedures by monitoring the results in the database. If the test results of a patient are not replicated at the same time at all sites, it is possible that inconsistent state of data might lead to a wrong decision being made. Online games applications also require that a move made by a player should be viewed by other players almost at the same time.

The problem of *Delay and Delay Variation Bounded Multicasting Network* (DVBMN) is one of finding a sub-network given a source and a set of destinations that satisfies the QoS (Quality of Service) requirements on the maximum delay from the source to any of the destinations and on the maximum inter-destination delay variance. Rouskas and Baldine [17] have shown that the DVBMN problem is NP-complete and have presented the first heuristic for the problem along with its performance evaluation. Kapoor and Raghavan [18] provided a novel heuristic that uses dynamic programming and showed that the delay variation obtained by their algorithm is significantly less than the heuristic of Rouskas and Baldine [17]. Unlike the heuristic of Rouskas and Baldine [17] which constructs a multicasting tree, the heuristic of Kapoor and Raghavan [18] constructs a multicasting network that may not be a tree. But on such a network no more than two messages are sent along each edge and hence it achieves the desired bandwidth savings. A simple heuristic was provided by Sheu and Chen [19], in which there are no bounds set on the delay variation. The complexity of

the heuristics proposed by Kapoor and Raghavan [18] and Rouskas and Baldine [17] is high when a new member joins the multicasting group or an existing member leaves the group.

In this research, we present a polynomial time heuristic for the DVBMN problem with the following characteristics:

- Our heuristic achieves the *tightest* (in a technical sense to be precisely defined later) possible bounds on delay variation along with Kapoor and Raghavan [18] but our heuristic outperforms the heuristic in [18] in terms of time-complexity. The complexity of our heuristic is $O(|E| + nk \log (|E|/n) + m^2k)$, where n and $|E|$ are the number of nodes and edges in the overlay network, respectively, m is the number of destinations and k is the number of shortest paths determined between source and destination nodes. The complexity of the heuristic in [18] is $O(\Delta|E|m\delta)$, which is pseudo-polynomial in delay and delay variation bounds, where m is the number of destination nodes and Δ and δ are the delay and delay variation bounds, respectively. Note that Δ is independent of the network size and hence even for smaller and less dense graphs, the heuristic in [18] requires large execution time. The time-complexity of the heuristic by Rouskas and Baldine [17] is $O(nmk(|E| + nk \log (|E|/n)))$[♦] using the best known k shortest path algorithm in [20]. Note that for dense graphs that is, $|E| = O(n^2)$ and $m = O(n)$ the time complexity of the algorithm in [17] is $O(n^4k + n^3k \log (|E|/n))$.

[♦] The complexity in [17] is shown to be $O(k^2mn^4)$ using a less efficient k-shortest path algorithm. We also implemented the heuristic of Rouskas and Baldine [17] using the faster k shortest path algorithm in [20].

For the same values of $|E|$ and m, our algorithm will have complexity of $O(n^2k)$. It should be pointed out that the heuristic by Rouskas and Baldine [17] does not provide the tightest delay variation as obtained by this research and Kapoor and Raghavan [18].

- We use an efficient k shortest path algorithm due to Victor and Andres [20, 21] on various network topologies with different edge densities. The k shortest path algorithm [20, 21] was executed on a 360MHz (SunSparc) computer. On a graph with 100 nodes with 80% edge density, 10 shortest paths between all pairs of nodes (10,000 of them) can be determined in just 15 seconds. Also, in order to find about 2,000 shortest paths between source and 10 destinations in a dense graph with 100 nodes it took only 65 seconds!
- Extensive simulations with varying number of nodes, edge densities, and size of multicasting group have shown that our heuristic outperforms heuristics in [17, 18] by a significant margin in terms of execution time while achieving the tightest possible delay variation.
- In order to perform *join* operation (where a single node joins the multicasting group) or *leave* operation (where a single node leaves the multicasting group) we show that our heuristic has a time-complexity of $O(m^2k)$ with the tightest delay variation. The heuristics in [17] and [18] have to be rerun in order to perform either a single join or leave operation, and hence incur complexities of $O(nmk(|E| + nk \log (|E|/n)))$ and $O(\Delta|E|m\delta)$, respectively.

1.5 Limitations of Traditional Multicasting Models

Overlay multicasting differs from traditional multicasting in several ways. Conventional multicasting network models do not capture all the behaviors of application level multicast. In any packet transmission, there are, in general, four types of delays, (i) processing delay (also known as nodal delay), (ii) queuing delay, (iii) transmission delay, and (iv) propagation delay. Of these, both nodal and queuing delays are associated with a node sending packets, and transmission and propagation delays are associated with a link. The *end-to-end delay* for packet transmission is the sum of all these delays between a source and a destination. In application level multicasting, an important nodal delay component (we will call this as the *duplicating delays*) occurs as a result of duplication and transmission of packets by an interior node in the multicast tree to its children in the tree. We will illustrate the concept of duplicating delays using an example of a multicasting tree on an overlay network.

Let us assume a multicast tree T on an overlay network with S as the root of the tree. Also assume that c_1, c_2, and c_3 are the children of S. Every node in the tree will use *sendTo* and *recvFrom* socket utilities [22] to send the packet that originated from S to its children in the tree and to receive the packet sent by its parent in the tree, respectively. Node S will execute *sendTo* three times, once for each of its children in the tree. Note that each of the send places the same size data on to the kernel buffer.

Now we have three copies of the same packet in the kernel send buffer and the UDP takes the segment (containing one data packet obtained as a result of the execution of *sendTo* function) and adds its header which is then passed to IP layer. The IP layer adds its header and places the packets in the data link layer queue. The frames in the queue (corresponding to each IP packet) are sent sequentially using both the logical link control protocol and the medium access control protocol. The medium access control layer transmits to the nearest router designated for the given host by gaining exclusive access to the channel and transmitting the frame. The delay experienced by the data link layer in sending a single frame is proportional to the channel access time and time required to receive the acknowledgement from the data link layer of the nearest router. This time on the average is the same for every frame sent by the host. The child node that receives information as a result of the second *sendTo* experiences additional delay due to the fact that the frames corresponding to the first *sendTo* have to be completed before its frame can be sent. Note that using current implementations of operating systems, applications may perform concurrent message transmissions. But this concurrent transmission will be serialized when it goes through a physical link [23].

Based on the discussion of the delays above, it is evident that the order in which the source S will issue the send to its children will decide when the children c_1, c_2, and c_3 will receive the packet from S. Let S send to the children in the following order, first to c_1, then to c_2, and finally to c_3. Let us assume that since S issues the send to c_1 first,

the additional delay experienced by it is 0 units. Let c_2 experience an additional delay of 3 units and c_3 experience of 5 units due to the fact that *S* sent the data packet using the second and third *sendTo* function statement executions at *S*, respectively. Generalizing this we will define a delay vector for a child node c_1 with two other siblings to be $< c_1^1, c_1^2, c_1^3 >$, where c_1^i, is the additional delay experienced when *S* sends the data packet to c_1 using the ith *sendTo* statement.

Given a multicast tree on an overlay network with link delays and duplicating delay vectors, we propose an algorithm which determines the order in which the data packets have to be sent by each non-leaf node to its children in the multicast tree such that the delay of the multicast tree is a minimum.

1.6 Embedding Multicasting Trees on the Overlay Network

Many collaborative applications require a multicasting tree for data delivery. These applications require that data be delivered within a desired time bound. To address this issue, we propose the problem of embedding a multicasting tree on the overlay network. An example of a tree network specified by an application designer is shown in Figure 1.3. The numbers of the links specify the maximum allowable delay on that

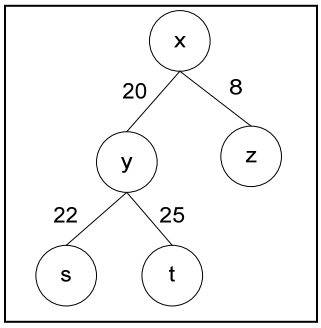

Figure 1.3: An example of a tree network specified by an application designer

link. The embedding problem is one of mapping nodes from a guest graph (the one the application designer desires) to a host graph (the overlay network) such that it maps (one-to-one) nodes in the guest graph to the nodes in the host graph and edges in the guest graph to the edges or paths in the host graph. We need to embed the multicasting tree on the overlay network in such a way that the embedding satisfies the delay constraint specified along each link in the tree.

Graph-Embedding problem for general graphs is defined as follows: given a graph (guest) $G = (V_G, E_G)$ and another graph (host) $H = (V_H, E_H)$, find an embedding function f such that f maps (one-to-one) nodes in G to nodes in H and f maps edges in G to paths or edges in H. Graph-Embedding problem for general graphs is a well known NP Complete problem [61]. In our multicasting tree embedding problem, the guest graph is a tree. Also, when we embed each edge of the multicasting tree on the overlay network, we want to ensure that the edge is embedded on a shortest path on the overlay network. To solve the multicasting tree embedding problem, we start with the

host graph as a shortest path tree. Now our problem is reduced to sub-tree embedding problem which is deciding whether a given tree can be embedded into another given tree.

A guest tree G_T can be embedded into a host tree H_T, if there exists an embedding function f (one-to-one) which maps nodes of G_T to nodes in H_T and edges of G_T to edges or paths in H_T. Different types of tree embeddings are defined depending on the conditions imposed on the embedding function f [25].

Isomorphic embedding: For every pair of nodes u_G and v_G of G_T, if there is an edge (u_G, v_G) in G_T, then there exists an edge from $(f(u_G) , f(v_G))$ in H_T.

Homeomorphic embedding: For every pair of nodes u_G and v_G of G_T, if there is an edge (u_G, v_G) in G_T, then there exists a path from $f(u_G)$ to $f(v_G)$ in H_T with all intermediate nodes of out-degree 1 and with no intermediate node coming from G_T.

In the multicasting tree embedding problem, an edge in the multicasting tree can be mapped to a path in the overlay network. Now our problem is similar to *sub-tree homeomorphism* problem. Given two trees G_T and H_T, the *sub-tree homeomorphism* problem is to find whether H_T has a sub-tree H_t that can be transformed into G_T by repeatedly removing any node of degree 2 and adding the edge joining its two neighbors. In multicasting tree embedding problem, we have to ensure that the embedding satisfies the delay constraint. In this research, we propose an algorithm for delay constrained multicasting tree embedding problem on the overlay network.

1.7 Organization of the Dissertation

The dissertation is organized in the following way. In chapter 2, we discuss about the floor control issues for collaborative application in overlay networks and provide an algorithm for constructing an efficient participating chain of NSNs. We provide the implementation of two distributed floor control protocols for overlay networks and compare them using an analytical model and simulation experiments. We also show that our implementation preserves causal ordering of messages. In chapter 3, we formally define the delay and delay variation bound multicasting problem and present a heuristic for the problem. Using simulation experiments, we show that our heuristic achieves better performance than other heuristics presented in the literature. In chapter 4, we illustrate the duplicating delay problem in overlay multicasting using an example. We also present an optimal scheduling algorithm for multicasting tree with duplicating delay vectors. In chapter 5, we formally define the delay constrained multicasting tree embedding problem on overlay network and propose an algorithm for the problem. Chapter 6 summarizes the contribution of this dissertation and outlines the direction and scope for future research.

Chapter 2

Floor Control in Collaborative Applications

2.1 Introduction

A distributed collaborative application can be formalized using a 3-tuple, $CA = (G, U, F)$ where $G = (V, E)$ denotes the network connecting the set of end-users V using the set of communication links E. F denotes the set of floors where each floor $f \in F$ is associated with a shared media (resource) used in the collaborative application. As the network links typically have non-negligible delays, let $d(e)$ denote the delay associated with a link $e \in E$. U represents the set of end-users participating in the collaborative application with $U \subseteq V$.

Floor Control Problem (FCP): Given a network $G = (V, E)$, a set of participating end-users U, and a set of floors F, design an access protocol on G which, at any instance of time, grants a floor $f \in F$ to only one user $u \in U$.

2.2 Overview of the solution

We adopt a two phased solution approach to solve the FCP. In phase one, we construct an efficient[2] communication channel $G' = (U, E')$ connecting the set of participating end-users U where $E' \subseteq E$. In the second phase, we design access control protocols on G' which will ensure that at any instance of time, a floor $f \in F$ is used by only one user $u \in U$. In this dissertation, for the sake of simplicity, we have discussed our implementation of floor control protocols for $|F| = 1$. However, we believe that our implementation of protocols can easily be extended to accommodate multiple floors in the system.

In an overlay network the end-hosts share the responsibility of creating, consuming, and forwarding messages to other end-hosts creating a virtual network architecture on top of the physical network. In the peer-to-peer architecture, the overlay functionality is pushed to the end-hosts, while in proxy-based architecture, the overlay functionality is provided through a set of distributed nodes called Network Service

[2] The efficiency of the communication channel can be measured as the maximum end-to-end delay between any pair of participating end-users in G'. In the next subsection, we will explain why an efficient communication channel is important for floor control protocols

Nodes (NSN). Most of the research work dealing with multicasting on overlay networks [1, 2, 3] use a proxy based architecture and in this dissertation too, we shall assume a proxy based architecture. In spite of there being differences between these two architectures, the issues regarding the floor control problem remain the same. Therefore, the proposed solutions are applicable to both proxy-based and peer-to-peer overlay networks. Thus, an NSN in our description that takes part in the floor control protocols will be referred to as an end-user in the peer-to-peer architecture. In a proxy based architecture, the NSNs communicate with end-hosts and with each other using standard unicast mechanisms.

Communication in any collaborative application in the overlay network can be divided into two phases: *contention phase* and *data delivery phase*. In the contention phase, an end-host that wishes to send data to other participating end-hosts through the floor informs its NSN of its intention. The NSN then contends with other such NSNs to acquire the floor and depending on the outcome, it sends either a grant signal or a rejection signal to the requesting end-host. In other words, contention phase is the phase during which the floor control problem occurs. Consequently, any FCP solution ought to be executed during this phase to resolve the contention. In the data delivery phase, the end-host, which was granted floor by its NSN in the contention phase, sends data to all other participating end-hosts through its NSN. In this dissertation, we present efficient protocols for both contention and data delivery phases.

2.2.1 Solutions for the Contention Phase

To overcome the deficiencies of the centralized approach as discussed in the previous chapter, decentralized approaches are considered in this research. Conventional MAC protocols resolve channel contention in a single-hop environment. Since FCP could be viewed as a channel contention problem in a multi-hop setting, MAC protocols present themselves as easily adaptable choices for solving the FCP. Towards this, we consider two well known MAC protocols: ALOHA and Distributed Queue Dual Bus (DQDB) and discuss efficient implementation strategies for these protocols on an overlay network. We consider ALOHA for floor control as it was the first randomized solution to channel allocation problem in MAC layer and is the basis of IEEE 802.3 standard protocol. The DQDB protocol is chosen for floor control as it is an elegant solution for a distributed network which achieves First In First Out (FIFO) order without having a centralized queue. Certain group communication applications such as video conferencing, collaborative design and simulation and online games require FIFO ordering of requests to maintain fairness among the users.

The effectiveness of most MAC protocols depends on the ability of a host to quickly detect the transmission from other hosts sharing the medium, which in turn, depends on the medium's propagation delay. When adopting MAC protocols as FCP solutions, care must be taken to minimize the delay involved in a host detecting the transmission of other hosts sharing the floor. Since a floor typically spans over a multi-

hop WAN, significant delays can occur in detecting a transmission of another host, especially when the other host is several hops away. Therefore, in order to increase the effectiveness of MAC protocols as FCP solutions, we first abstract the WAN connecting the NSNs as a communication channel. The constructed channel is such that it minimizes the maximum end-to-end delay between any pair of NSNs in the overlay[3]. Once the channel abstraction is done, the MAC protocols are implemented on top of this channel to solve the FCP.

2.2.2 Solutions for the Data Delivery Phase

As mentioned earlier, in the data delivery phase, the end-host which was granted floor by its NSN in the contention phase sends data to all other participating end-hosts through its NSN. Collaborative applications require that data sent by end-hosts be received by other participating end-hosts in the order of floor acquisition (for e.g. in online games, players want to see the moves made by other players in proper order). If an end-host A acquires the floor before another end-host B, then all the participating end-hosts should receive the data sent by end-host A before they receive the data sent by end-host B. This implies that *causal ordering* [27, 28] of messages should be maintained in the data delivery phase. Causal ordering guarantees that messages are delivered to the destinations according to their sending order. As the participating end-

[3] Other metrics of efficiency such as average waiting time at the intermediate NSNs and loss probability can also be used to construct this communication channel.

hosts are geographically distributed, causal ordering in the data delivery phase does not occur automatically and additional mechanisms are required for enforcing the same. In this research, we present algorithms employing which causal ordering of messages can be ensured in the data delivery phase.

2.3 Optimal Participating Chain for Contention Phase

As discussed earlier, the contention phase involves two tasks. The first task is to construct an efficient communication channel that connects all the NSNs and the second phase is to develop contention resolution protocols that employ this communication channel to allocate the floor for the interested NSNs. This section presents the details of the first task on constructing an efficient communication channel.

We call an NSN to be *participating*, if there is at least one participating end-user attached to it. We aim to construct an efficient communication channel by creating a chain of the participating NSNs such that the resulting chain has the least end-to-end delay among all possible chains.

Consider a weighted graph of NSNs, $G = (V, E)$, a set of participating NSNs $P \subseteq V$ and a non-negative delay $d(e)$ for each edge $e \in E$. A *participating chain* $P_C = (V_C, E_C)$ where $P \subseteq V_C \subseteq V$ and $E_C \subseteq E$ is a path (simple or non-simple) that connects the

set of participating NSNs P. For example an Euler path can be a participating chain. The length of a participating chain is defined as $\sum_{l \in P_C} d(l)$. The participating chain can be thought of as a token-bus network where the maximum end-to-end delay is equal to the length of the chain. Considering the link delays in the overlay network, this end-to-end delay could be large depending on the choice of the participating chain.

Problem Optimal Participating Chain: Given a graph $G = (V, E)$, a set of vertices $P \subseteq V$, a delay function $d(e)$ for each edge $e \in E$, the problem is to construct a path P_C (simple or non-simple) that connects all the vertices in V such that, delay $(P_C) \leq$ delay (P') for all possible P'.

If G is un-weighted, then *Hamiltonian walk* on G is the optimal participating chain [29]. If G is weighted, then *Hamiltonian walk* on G may not be the optimal participating chain. For example, consider the graph G on Figure 2.1. The *Hamiltonian walk* on G is A-E-B-C-D with length 6+2+10+4 = 22. The optimal participating chain on G will be A-E-B-E-D-C with length 6+2+2+2+4 = 16. Given a weighted connected graph, we are interested to find the optimal participating chain.

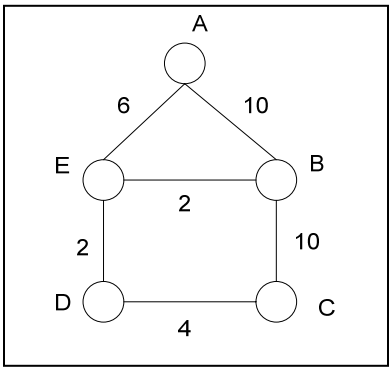

Figure 2.1: A graph G = (V, E)

We develop an algorithm to obtain an optimal participating chain for a tree. Given a weighted graph, a Steiner tree can be constructed using the efficient heuristics presented in the literature [56, 65]. Our algorithm takes a Steiner tree as an input and produces an optimal participating chain for the Steiner tree. Before we explain our algorithm, we list down the properties of an optimal participating chain of a tree network.

Property 1: An optimal participating chain (P_C) of a tree network will start at a node of degree 1 and end at a node of degree 1.

Proof: Suppose we have a tree T with $k + 1$ nodes and root p. Clearly p has k children. Let the children of p be $c_1, c_2, c_3, \ldots, c_k$. Let us assume that w is an optimal P_C on T and that w starts from p which has degree $k > 1$. Then w will be p - c_1- p - c_2 – p - c_3 – p- … - p - c_k. Let w' be another P_C which starts from c_1. Then w' will be c_1 – p - c_2 – p - c_3- … - p - c_k. It is easily observed that length(w) = length (w') + 1. So w cannot be the optimal P_C on T. This is a contradiction. So an optimal P_C on a tree will start with a

node of degree 1. In a similar way, we can prove that an optimal P_C on a tree will end with a node of degree 1. ■

Property 2: An optimal participating chain (P_C) of a tree network will not traverse the links in the longest path of the tree more than once.

Proof: Let us assume that the weight of link c_1 - p in T is lw_1, weight of link c_2 - p in T is lw_2,…, weight of link c_k - p in T is lw_k. Without loss of generality we can also assume that $lw_1 > lw_2 > lw_3 > \ldots > lw_k$. Note that the longest path in T will be $c_1 - p$ - c_2. Suppose w_1 is an optimal P_C on T which traverses the links of the longest path more than once. Let w_1 be $c_1 - p$ - $c_2 - p$ - c_3 - … - p - c_k. Suppose w_2 is another P_C chain (which traverses c_2 at the end) $c_1 - p$ - $c_3 - p$ - c_4 - … - p - $c_k - p$ - c_2. It can be easily observed that length (w_1) > length (w_2). So w_1 is not the optimal P_C of T. This is a contradiction. So the optimal P_C of a tree will not visit the links of the longest path more than once. ■

Property 3: Different order of visiting the sub-trees of children (of the root) which are not contained in the longest path does not change the length of the optimal P_C.

Proof: Let us assume that c_1 and c_2 are the children of p which are contained in the longest path of T. Let w_1 be an optimal P_C which starts at c_1 and ends at c_2. So w_1 will be $c_1 - p$ - $c_3 - p$ - $c_4 - p$ - c_5 - … - p - $c_k - p$ - c_2. Let w_2 be another P_C which starts at c_1 and ends at c_2 but visits the other children of p in different orders. Let w_2 be $c_1 - p$ -$c_4 -$

p - c_5 – p - c_3 – p - … - p - c_k – p - c_2. It is easily observed that length (w_1) = length (w_2). This implies that optimal P_C of a tree is independent of the ordering of children (or the sub-trees rooted at the children) of the root which are not contained in the longest path. ■

In order to derive the optimal P_C from tree T, first we determine the root of the T such that root is the center of the tree. Center location on a tree network is a well known problem and several algorithms have been presented in the literature [30-34]. If there are more than one center nodes in T, we choose one of them arbitrarily. Suppose the center location algorithm returns r as the root of T. We rearrange T with respect to r.

Now we want to traverse (visit all the nodes at least once) T so that the length of the traversal (chain) is minimum. The process of finding the chain is a three step process: (a) identifying the nodes in the longest path in the rooted tree T, (b) rearranging the children of non-leaf nodes (which are in the longest path) in the tree T, and (c) performing an in-order traversal of the rearranged tree.

Let u and v be the two children of r which are contained in the longest path of T. Let $\{x_1, x_2, \ldots, x_k\}$ be the set of leaf nodes of the subtree rooted at x in the tree. Also, let us denote the path length from node x to the leaf node x_i by $l(x, x_i)$ and let $l(x) = max\ \{l(x, x_i), 1 \le i \le k\}$. We order the children of r in non-increasing order of their l values calculated as above. The first and second node in the sorted order will be named as v

and u, respectively. We rearrange the children of r such that u and v are the leftmost and rightmost child, respectively. Then we rearrange the children of u (respectively v) in the decreasing (respectively increasing) order of their l values. Now the in-order traversal of the new tree will produce the desired chain. We will use the following notations and definitions as part of our algorithm.

$T(r)$: Tree with root r.

$ST(c)$: Subtree of T rooted at c.

Ordering Symbol: $\{\leq, \geq\}$.

Order ($T(r)$, < *ordering symbol* >): Arrange k children of r in T such that $l(c_i)$ < *ordering symbol* > $l(c_{i+1})$ and c_i is a child of r for $1 \leq i \leq k$.

Algorithm *find_chain_*P_C ( )

Input: $T(r)$

Output: chain P_C

Begin

Compute $l(x)$ for each non-leaf node x in T.

Sort the children of r in non-increasing order of their l values.

Let v and u be the first and second child of r in the sorted order.

Rearrange T such that u and v are the first and last child of r.

For each non-leaf node i in $ST(u)$

Order ($ST(i)$, ≥).

End For

For each non-leaf node l in the $ST(v)$

Order $(ST(l), \leq)$.

End For

P_C = In-order traversal on T.

End

Figures 2.2, 2.3 and 2.4 illustrate the algorithm using an example.

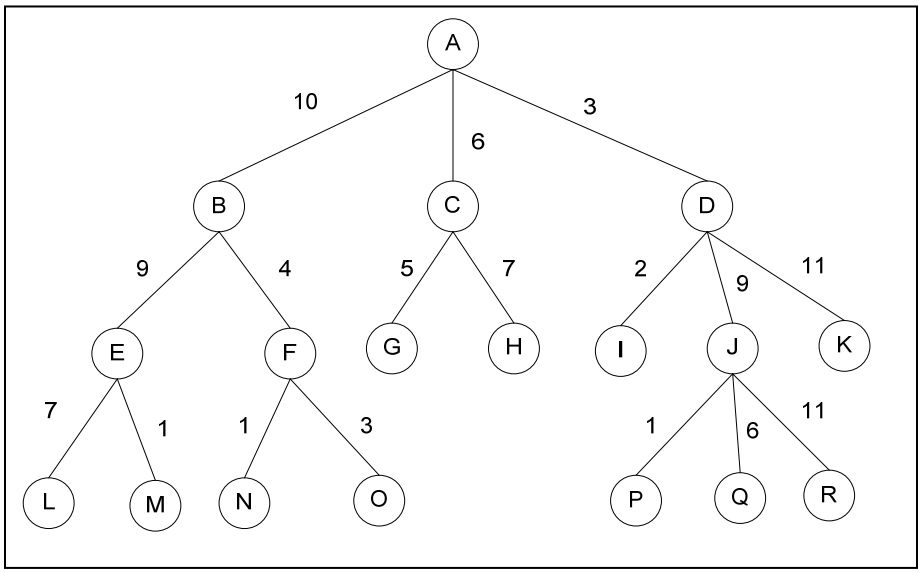

Figure 2.2: A tree with node A as root

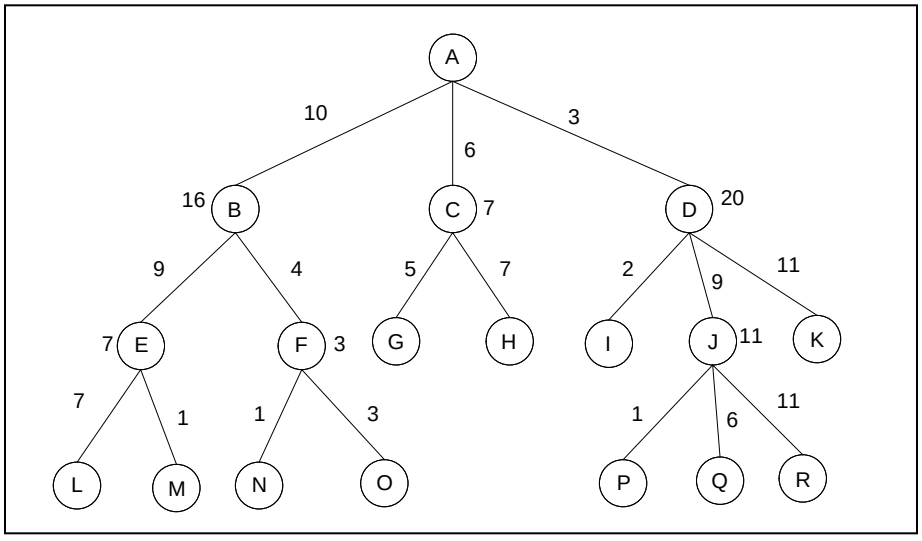

Figure 2.3: The tree after bottom-up computation of $l(x)$ for each non-leaf node x

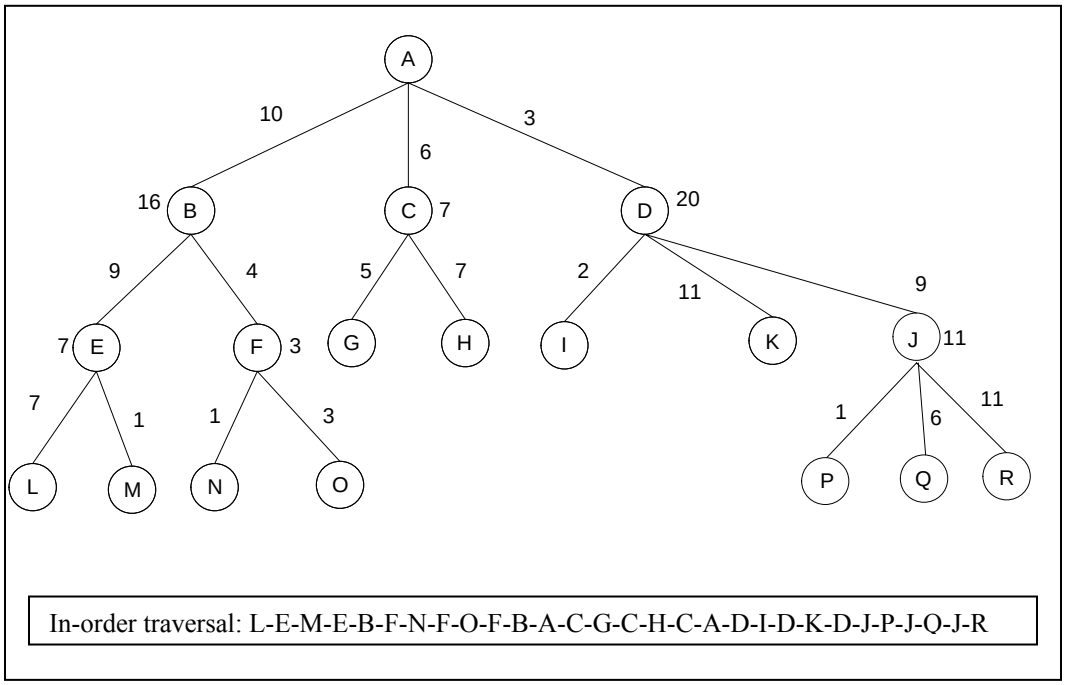

Figure 2.4: The rearranged tree with the in-order traversal

Using the properties of optimal chain of a tree, the following two theorems can be easily proved.

Theorem 2.1: Algorithm *find_chain_*P_C will always return the optimal traversal on $T(r)$.

Proof: According to property 2, the optimal traversal of a tree will not visit the links of the longest path of the tree more than once. In our algorithm, first we determine the children of the root *r* which are contained in the longest path. Then we rearrange the tree in such way that the children of *r* which are contained in the longest path are

visited only once in the in-order traversal of *T*. The first *For* loop in the algorithm ensures that one of the end-nodes in the longest path is made the leftmost leaf node in the rearranged tree and the second *For* loop ensures that the other end-node in the longest path is made the rightmost leaf node in the rearranged tree. This implies that in the rearranged tree, the path from the leftmost child to the root and the path from root to the rightmost child constitute the longest path of the tree. The in-order traversal will ensure that the links in the longest path of the tree are visited only once. It also ensures that the traversal starts with a node of degree 1 and ends with a node of degree 1 (Property 1). ■

Theorem 2.2: Algorithm *find_chain_*P_C has a complexity of $O(nlogn)$.

Proof: A tree with n nodes has $O(n)$ edges. Initially we sort the edges of the tree (which takes $O(nlogn)$) and store the sorted edges in an array. Each non-leaf node keeps pointers to the elements in the sorted array which correspond to its children. This can be done by scanning the sorted array only once. The Order ( ) function in the algorithm rearranges the children of a non-leaf node based on the pointers to the sorted array. ■

2.4 Protocols for Resolving Floor Contention

Once the communication channel has been constructed, the participating NSNs employ contention resolution protocols to gain access to the floor. The contention resolution protocols can either be randomized or scheduled. In randomized protocols, the NSNs try to detect collision and either grant or reject the floor requests of the attached end-hosts depending on the outcome of the collision detection mechanism. If an end-host receives floor rejection message, it waits for random amount of time and sends the floor request again to its NSN. In the case of scheduled protocols, each NSN that has received a floor request from its end-host, waits for its turn as in a token passing mechanism. This mechanism can be round robin or first-in-first-out. In this research, we consider ALOHA and CSMA protocols for the randomized type and DQDB protocol for the scheduled type. The generic protocol for floor acquisition at the end-host is as follows.

Protocol End-host_floor_acquisition( )

Begin

If end-host wants to acquire floor

Generate *floor_request* message

Send *floor_request* message to NSN

End If

If end-host receives *floor_grant* message from NSN

End-host goes to *Data Delivery Phase*

Else If end-host receives *floor_reject* message from NSN

End-host waits for random amount of time

End-host_floor_acquisition( )

End If

End

2.4.1 ALOHA for Floor Control

In the conventional ALOHA protocol [23], nodes send data whenever they are ready. When two nodes send data at the same time, a collision occurs and the colliding frames are garbled. The sending node can find out whether its frame is garbled by listening to the channel. If the frame is garbled, the sending node waits a random amount of time and sends it again. Implementing ALOHA protocol for floor control on the overlay network poses a challenge in that a technique is required for the NSNs to detect the collisions of request signals. We propose a technique wherein, an NSN waits for certain period of time after forwarding the floor request signal (received from its end-host) to other NSNs. During the waiting period if it receives another floor request signal, it determines that a collision has occurred. An important parameter which

dictates the efficiency of the above floor access solution is the duration of the waiting period which in turn depends on the length of the participating chain[4].

Every NSN in the participating chain (P_C) maintains a list of *neighbor NSNs* < *l_nbor*, *r_nbor* >, where *l_nbor* and *r_nbor* are the IDs of the left and right side neighbors in the chain respectively. For the first and last NSN in P_C, the *l_nbor* and *r_nbor*, respectively will be null. We assume that every NSN knows the length of P_C (say *C*). When an end-host wants to acquire the floor, it sends a *floor_request* signal to its NSN. The NSN forwards the *floor_request* signal to its *neighbor NSNs* and waits for 2*C* amount of time period. The neighbor NSNs then forward the *floor_request* to their left or right neighbors, as the case may be, and the process continues. If the originating NSN does not receive any forwarded *floor_request* signal from its *neighbor NSNs* during the waiting period, it assumes that there is no contention for the floor. It then grants the floor to the requesting end-host by sending a *floor_grant* signal. The requesting end-host, then enters into the data delivery phase (discussed in Section 2.5) for sending data to all the participating end-hosts. If the NSN, which forwarded the generated *floor_request* signal, receives *floor_request* signals from its *neighbor NSNs* during the waiting period, it figures out that its end-host's *floor_request* signal is colliding with the *floor_request* signal of at least one other end-host and rejects the *floor_request* by sending a *floor_reject* signal to the requesting end-host. The

[4] It is for this reason, we seek to construct a participating chain that minimizes the maximum end-to-end delay.

requesting end-host then backs off and waits for random amount of time and resends its *floor_request* signal to its NSN.

The length of the waiting period is set to 2*C* because in the worst case, the end-host attached to the first NSN in P_C will be contending for floor against the end-host attached to the last NSN in P_C. When the last NSN in P_C forwards a *floor_request* signal, it will take *C* units of time to reach the signal at the first NSN. If the first NSN in P_C waits for 2*C* units of time after it has forwarded the *floor_request* (which was generated by its end-host) to its *r_nbor*, the NSN will be able to detect whether the request is colliding with a request generated by an end-host attached to the last NSN. The protocol is as follows.

NSN_ALOHA_on_P_C

Begin

If NSN receives *floor_request* from its end-host

floor_control (*floor_request*)

End If

If NSN receives *floor_request* from *l_nbor*

NSN forwards *floor_request* to *r_nbor*

End If

If NSN receives *floor_request* from *r_nbor*

NSN forwards the *floor_request* to *l_nbor*

End If

End

floor_control (*floor_request*)

Begin

Send *floor_request* to *l_nbor* and *r_nbor*

Set *timer* = 2*C*

While *timer* ≠ 0

If *floor_request* is received from *l_nbor* OR *r_nbor*

Send *floor_reject* to end-host

Break

End If

timer--

End While

Send *floor_grant* to end-host

End

The protocol can be described using the finite state machine of Figure 2.5 where state space S = {IDLE, PASSIVE, CONTENDING, GRANT, and REJECT}. Transitions are triggered when an NSN receives a message or the timer expires. Initially an NSN is in the IDLE state. When an NSN receives a *floor_request* signal from one of its *neighbor NSNs*, it goes to PASSIVE state. In this state, the NSN forwards the *floor_request* signal to its other neighbor and changes its state back to IDLE. When an NSN receives a *floor_request* signal from its end-host, the NSN goes

to CONTENDING state. In the CONTENDING state, the NSN sets its timer to 2*C*. If another *floor_request* is received in this state, the NSNs goes to REJECT state. In this state, the NSN sends *floor_reject* signal to the requesting end-host and changes its state back to IDLE state. When the NSN is in the CONTENDING state, if the timer expires and no *floor_request* signal is received, the NSN switches to GRANT state. In the GRANT state, the NSN sends *floor_grant* signal to the requesting end-host and changes its state back to IDLE state.

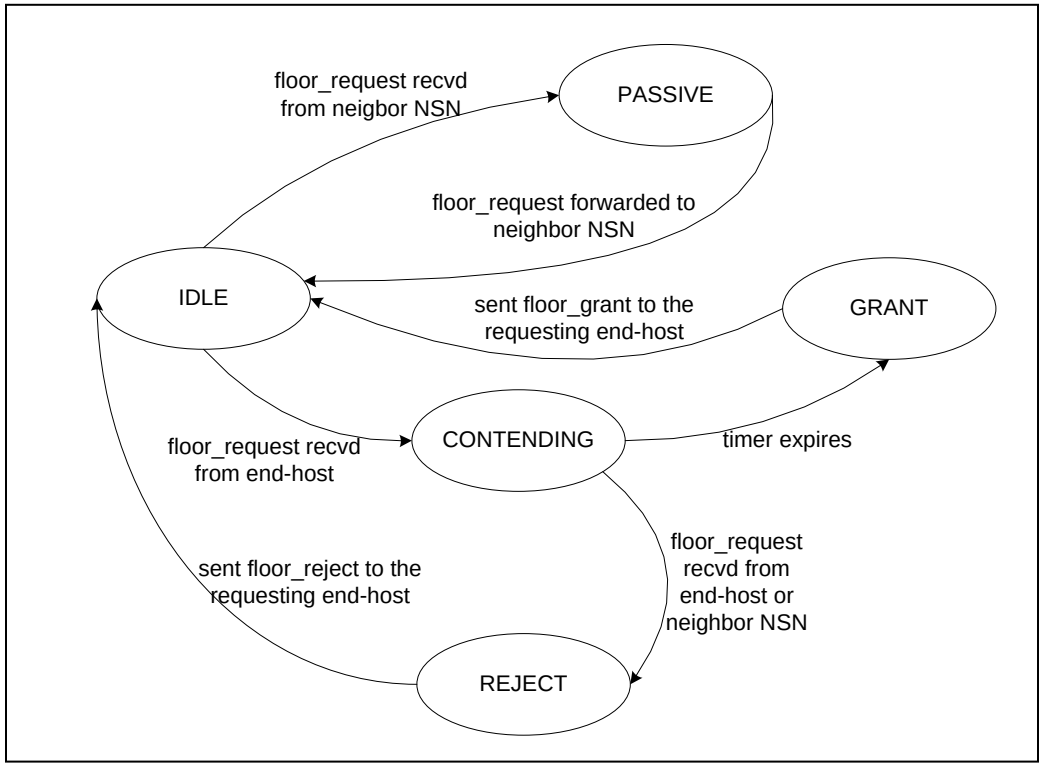

Figure 2.5: State Diagram for ALOHA on a participating chain

2.4.2 DQDB for Floor Control

IEEE 802.6 protocol [23] uses Distributed Queue Dual Bus (DQDB) for Metropolitan Area Network (MAN). In this protocol, stations are connected to two parallel unidirectional buses. Each bus has a head-end and a steady stream of cells is generated from each head-end. A cell travels through the bus and when it reaches the end, it just falls off. When a station wants to send data to another station, it has to know whether the destination station is on its left or right. If the destination is to the right, the sender uses (say) Bus 1 for sending request signals and (say) Bus 2 for sending data. If the destination is to the left of the sender, the roles of Bus 1 and Bus 2 are inter-changed.

We can implement DQDB for floor control by arranging the participating NSNs in a chain called participating chain (P_C) using the algorithm mentioned in section 2.3. Two unidirectional logical paths are maintained among the NSNs: *request_flow* and *signal_flow* path as shown in Figure 2.6.

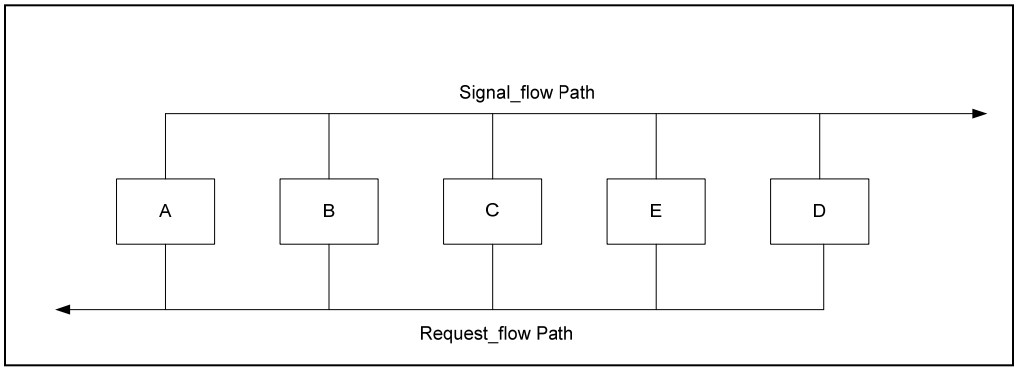

Figure 2.6: A participating chain with request_flow path and signal_flow path

In Figure 2.6, the request_flow path is D-E-C-B-A and the signal_flow path is A-B-C-E-D. The request_flow path carries requests in one direction and the signal_flow path carries signals in the opposite direction. The first NSN in the signal_flow path is known as *head_NSN* and the first NSN in the request_flow path is known as *tail_NSN*. Each NSN knows its *neighbors* < *l_nbor*, *r_nbor* > in each path. Each NSN also maintains two variables: *RQ* and *DQ*. *RQ* of an NSN keeps track of the number of requests forwarded by the preceding NSNs in the request_flow path before the NSN gets its chance to acquire the floor for its end-host. *DQ* of an NSN denotes the position of NSN's end-hosts' request in the distributed queue. Initially the *DQ* value and the *RQ* value of each NSN are assigned to 0.

When an end-host wants to acquire the floor, it sends a *floor_request* signal to its NSN. The NSN of the requesting end-host forwards the request signal to its *l_nbor* in the request_flow path. The NSN of the requesting end-host also copies its *RQ* value to its *DQ* value and resets its *RQ* to 0. The *l_nbor* NSN of the requesting NSN (we call the NSN attached to the requesting end-host as requesting NSN) increments its *RQ* value when it receives the *floor_request* signal in the request_flow path and forwards the signal to its *l_bor* in the request_flow path. All the downstream NSNs in the request_flow path, after receiving the *floor_request* signal, increment their *RQ* values and forward the request to their *l_nbors* in the same path. An agent program at the *head_NSN* generates a *wake_up* signal which propagates through the signal_flow path until it reaches the *tail_NSN*. Each *wake_up* signal carries a sequence number which is incremented each time the signal is generated. Before forwarding *wake_up* signal to its

r_nbor in the signal_flow path, the *head_NSN* checks its *DQ* value. If its *DQ* value is 0, the *head_NSN* checks whether it has any requesting end-host. If it has a requesting end-host, the *head_NSN* accepts the *wake_up* signal and sends *floor_grant* signal to its requesting end-host. In this case, the *head_NSN* puts the end-host ID in the *wake_up* signal and forwards the signal to its *r_nbor* in the signal_flow path. If the *DQ* value of the *head_NSN* is greater than 0, the *head_NSN* decrements its *DQ* value and forwards the *wake_up* signal to its *r_nbor* in the signal_flow path without putting the ID of the end-host in the signal. When an NSN receives a *wake_up* signal from its *l_nbor* in the signal_flow path, it checks whether the signal contains any end-host ID (which implies that the *wake_up* signal is already used). If the signal contains an end-host ID, it just forwards the signal to its *r_nbor* in the signal_flow path. If the signal does not contain any end-host ID, the NSN checks its *DQ* value. If its *DQ* value is 0, the NSN knows this is its end-host's turn to acquire the floor. So the NSN puts the end-host ID in the *wake_up* signal, forwards the signal to its *r_nbor* in the signal_flow path, and sends *floor_grant* signal to its requesting end-host. If the *DQ* value of an NSN is greater than 0, the NSN just decrements its *DQ* value and forwards the *wake_up* signal to its *r_nbor* in the signal_flow path. The *head_NSN* generates *wake_up* signal periodically. The DQDB protocol on participating chain for floor control is given below.

NSN_DQDB_on_P_C ( )

Begin

If NSN receives *floor_request* from its end-host

Forward *floor_request* to *l_nbor* in request_flow path

DQ ← *RQ*

RQ ← 0

End If

If *floor_request* is received in request_flow path

RQ ← *RQ* + 1

Forward *floor_request* to *l_nbor* in request_flow path

End If

If *wake_up* is received in signal_flow path

If *wake_up* does not contain end-host ID

If (*DQ* = 0)

Send *floor_grant* to requesting end-host

Put end-host ID in the *wake_up* signal

Forward the *wake_up* signal to *r_nbor* in signal_flow path

Else

DQ ← *DQ* - 1

Forward *wake_up* to *r_nbor* in signal_flow path

End If

Else

Forward *wake_up* to *r_nbor* in signal_flow path

End If

End If

End

The state diagram of the protocol is given in Figure 2.7. In this state machine the state space S = {IDLE, PROECESS GENERATED REQUEST, PROCESS RECEIVED REQUEST, PROCESS WAKE-UP SIGNAL and GRANT}. Transitions are triggered when an NSN receives a *floor_reuest* from its end-host or receives a forwarded *floor_request* from its neighbors or receives a *wake_up* signal. Initially an NSN is in the IDLE state. When an NSN receives a *floor_request* signal from its end-host, it goes to PROCESS GENERATED REQUEST state. In this state, the NSN modifies its state variables (*DQ* and *RQ*), forwards the request to its appropriate *neighbor* and goes back to IDLE state. When an NSN receives a forwarded *floor_request* signal from its neighbor NSN in the request_flow path, it goes to PROCESS RECEIVED REQUEST state. In this state, the NSN modifies its state variables, forwards the request to its appropriate *neighbor* and goes back to IDLE state. When an NSN receives a *wake_up* signal, it changes its state to PROCESS WAKE-UP SIGNAL state. In this state, the NSN checks its *DQ* value. If the *DQ* value is 0 and the NSN has previously requested end-host, the NSN goes to GRANT state. In the GRANT state, the NSN sends *floor_grant* signal to the requesting end-host.

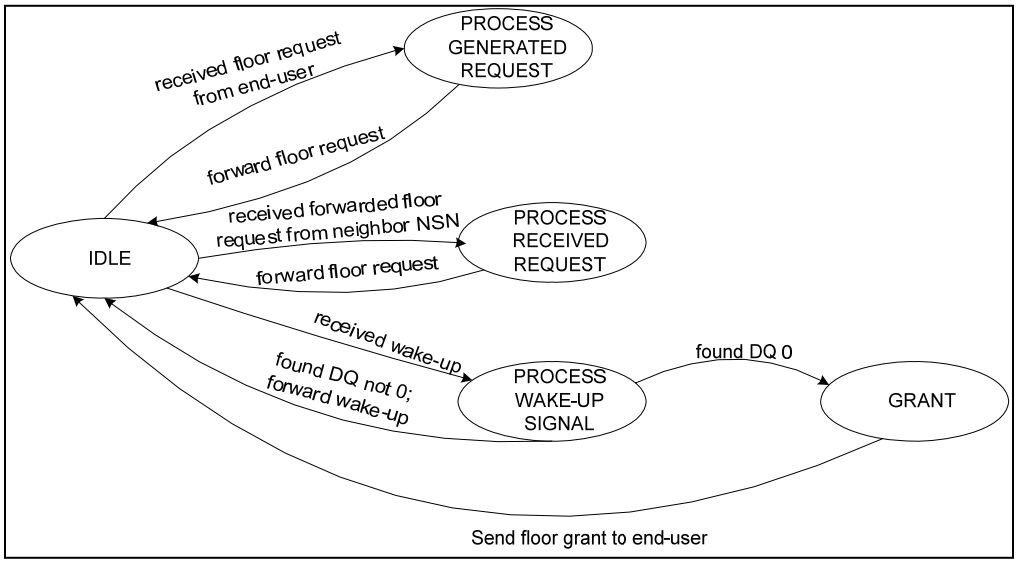

Figure 2.7: State Diagram for DQDB on a participating chain

Note that in our participating chain, an NSN can appear more than once. For example in the participating chain of Figure 2.8, node NSN *C* appears twice. In this situation, one of them will work as primary NSN and others will work as secondary NSNs. In the chain of Figure 2.8, NSN *C* first appears between NSN *B* and *D* and then appears between NSN *D* and NSN *E*. We can assign the first one as primary NSN and the second one as secondary NSN. A secondary NSN is not attached to any end-user, it just forwards *floor_request* and *wake_up* signals along the two paths.

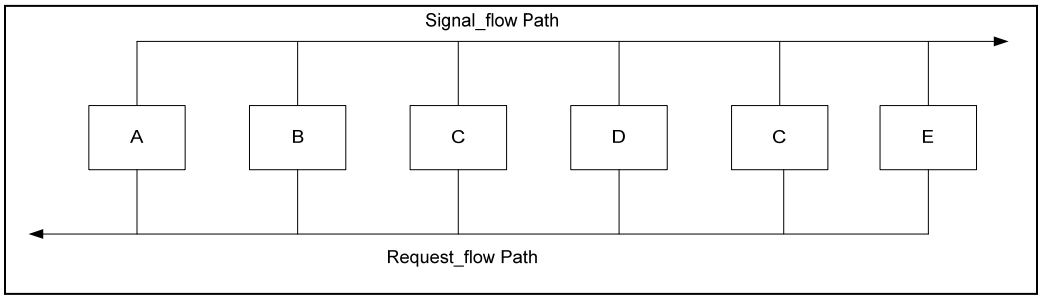

Figure 2.8: A participating chain where an NSN appears more than once

2.5 Data Delivery Phase

In the contention phase, the end-hosts contend for floor acquisition through their attached NSNs. The end-host which acquires the floor enters into *data delivery phase* for sending data to other participating end-hosts. We propose to use the participating chain and the optimal multicasting tree for data delivery. Using the participating chain we want to ensure that causal ordering of messages is maintained among the participating end-hosts.

The optimal multicasting tree can be a single source shortest path tree when we want to minimize the maximum end-to-end delay in data delivery. If the collaborative application requires that data should be reached at all participating end-hosts almost at the same time, then we can use the multicasting tree with minimum end-to-end delay and minimum delay variation [17-35] as the optimal multicasting tree. We connect the root of the multicasting tree to the last NSN (*tail_NSN* in case of DQDB) in the participating chain and show that it can achieve causal ordering.

2.5.1 Data Delivery Phase for ALOHA

When an end-host (which acquires the floor) enters into data delivery phase, it sends data to its NSN. The NSN forwards the data along the participating chain until it reaches the last NSN in the chain. The last NSN in turn, forwards this data to the root

of the multicasting tree which forwards it to all the participating end-hosts. Note that when the NSNs forward data to the root of the multicasting tree, they do not forward data to their attached participating end-hosts. To prove that the proposed implementation of ALOHA maintains causal ordering, let us consider the following worst case scenario. NSN A and B are the first and last NSNs of the participating chain respectively. At time $t = 0$, A forwards the floor request of its end-host to its neighbor in the chain (Figure 2.9). Then A waits for $2C$ time period. Assume that A does not receive any floor request during the waiting period. So A will grant floor to its requesting end-host at time $t = y$ where $y = 2C$. Then the end-host will send data to A and A will forward the data to the root of the multicasting tree. It will take $C + \delta$ units of time to reach the data from A to the root where δ is the path delay between B (which is the last NSN) and the root. Now consider that B forwards a floor request (generated from its end-host) at time $t = x$. Clearly $x > C$, otherwise A would have received the floor request forwarded by B which could have resulted in a floor rejection at A. B waits for $2C$ time period. Assume that B does not receive any floor request during the waiting period. So B will grant floor to its end-host at time $t = z$ where $z = x + 2C$. Then the end-host sends data to B and B forwards the data to the root of the tree. It will take δ units of time to reach the data from B to the root. We have to prove that the root receives the data of A before B. Let u and v be the values of t when the root receives the data from A and B respectively. Therefore, $u = y + C + \delta = 3C + \delta$ and $v = z + \delta = x + 2C + \delta$ It can be easily observed that $v > u$ as $x > C$. So our implementation guarantees that causal ordering will be maintained among the participating end-hosts.

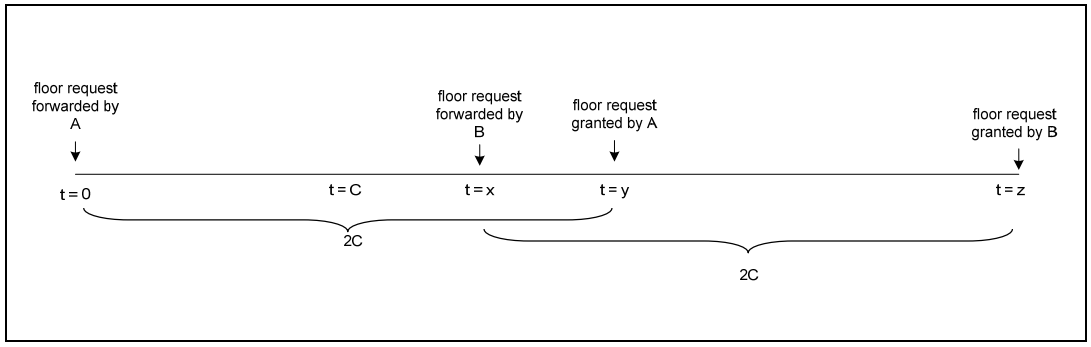

Figure 2.9: Timing diagram for floor control signal by NSNs A and B in ALOHA

2.5.2 Data Delivery Phase for DQDB

In case of DQDB, when the *tail_NSN* receives the *wake_up* signal in the signal_flow path, it forwards the signal to the root of the multicasting tree. The root keeps the sequence number of the *wake_up* signal in the database. If the *wake_up* signal does not contain an end-host ID, then the root assumes that the *wake_up* signal was not used by any end-host. If the signal contains an end-host ID, the root keeps both the sequence number and the end-host ID in the database. When an end-host receives *floor_grant* signal from its NSN, it enters into data delivery phase. In this phase, the end-host sends data along with its ID to its NSN. The NSN forwards the data along the signal_flow path until the data reaches the *tail_NSN*. Then the *tail_NSN* forwards data to the root of the multicasting tree. Now the root checks the ID of the end-host in the database. If the sequence number associated with the end-host is the smallest among the sequence numbers (associated with end-host IDs), the root forwards the data of the end-host to other participating end-hosts using the multicasting tree and removes the entry

(sequence number and its end-host ID) from the database. Otherwise the root buffers the data until it receives the data from the end-host with the smallest sequence number in the database (as shown in Figure 2.10). This implementation ensures that all the participating end-host receives data according to the order of floor acquisition which in turn ensures causal ordering of data among the participating end-hosts.

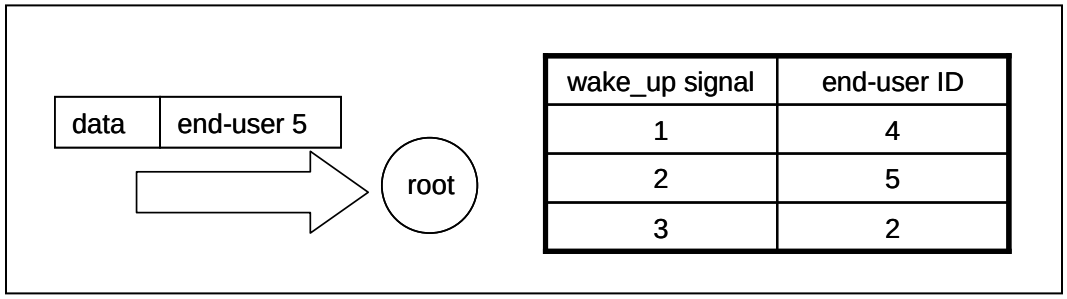

Figure 2.10: Buffering end-user data at the root for DQDB

2.6 Analytical Evaluation

In order to compare the described protocols, we use efficiency as the performance metric. In this section we derive the efficiency of each protocol using the analytical model defined in [11] as the basis. We also consider the overlay network delays in the evaluation. Our intention is to compare each protocol considering the underlying communication infrastructure in the contention phase. Efficiency of a protocol is defined as the ratio of floor usage time and turn around time. Floor usage time denoted by *U* is the activity time of an end-host when it uses the floor to send messages to other end-hosts. Turn around time denoted by *T* consists of two components: contention time

(X) and floor usage time (U). The contention time, X accounts for the time required for request generation and propagation and waiting period before acquiring the floor. So we have,

$$T = X + Y$$

$$\text{Efficiency, } \eta = \frac{U}{T} = \frac{U}{X + U} \quad \ldots \quad \ldots \quad \ldots \quad (1)$$

To derive the expression of contention time, X, we consider the worst case scenario for each protocol using the communication infrastructure in the contention phase. We assume that control messages in the protocol are transferred reliably over the network. The floor request generation rate follows Poisson distribution. The parameters used in the evaluation are listed in Table 2.1.

Parameter	Description
γ	Average processing time of control message at each NSN
δ	Duration of average activity period for each end-host
η	Efficiency of the protocol
λ	Floor request inter-arrival rate for end-hosts
σ	Average delay between an end-host and its NSN
n	Number of participating NSNs in the group communication
α	Frequency of wake-up signals generated by head NSN
C	Total Delay of the Communication Chain

Table 2.1: Parameters used in the analysis

γ is the average time to process and propagate a control message at each NSN. δ is the average time used by each end-host for sending data to other end-hosts when it gets the floor. λ denotes the inter-arrival rate for floor requests. n is total number of *participating* NSNs participating in the group communication. α is the frequency of wake-up signal generation by the agent program at head NSN in DQDB. C denotes the total delay of the communication chain.

2.6.1 Efficiency of ALOHA on a Participating Chain

The efficiency of pure ALOHA protocol as derived in [23] is

$$\eta = G e^{-2G} \quad \ldots \quad \ldots \quad \ldots \quad (2)$$

where G is the total rate of transmission attempts per frame time. When we implement ALOHA on the participating chain, we assume that all the NSNs are actively monitoring each other's activity. In the worst case analysis, when an end-host connected to the leftmost NSN in the participating chain generates a floor request signal, it will take $\sigma + C_1 + C_2 + \ldots \ldots + C_{n-1}$ units of time to reach the signal at the rightmost NSN of the chain. Each NSN takes γ units of time to process and propagate the control message. So the total overhead in communication is $\sigma + C + n\gamma$ where $C = C_1 + C_2 + \ldots \ldots + C_{n-1}$. The vulnerability period of a packet is defined [23] to be the time period if another packet is generated during that time period, the two packets will collide. In the pure ALOHA implementation on a participating chain, the floor request of an end-host is not going to collide with floor request from other end-hosts if no

requests are generated during the vulnerability period of $2(\delta+\sigma + 2C + n\gamma)$. The request arrival rate is Poisson. So the probability that a floor request is successful is $e^{-2(\delta+\sigma + 2C + n\gamma)\lambda}$. Using Equation 2, we derive the efficiency of ALOHA on a participating chain as follows:

$$\eta = (\delta + \sigma + 2C + n\gamma)\lambda e^{-2(\delta+\sigma+2C+n\gamma)\lambda} \quad \ldots \quad \ldots \quad \ldots (3)$$

2.6.2 Efficiency of DQDB on a Participating Chain

When we implement DQDB on a participating chain, the first NSN on the signal_flow path and request_flow path are designated as *head_NSN* and *tail_NSN* respectively. In the worst case analysis, when an end-host attached to the *tail_NSN*, sends a request and the *tail_NSN* forwards the request in the request_flow path, it takes $(\sigma+ C + n\gamma)$ units of time to reach the request to *head_NSN*. Since the floor request arrival rate is λ, the number of requests generated during the time period $(\sigma+ C + n\gamma)$ will be $(\sigma+ C + n\gamma)\lambda$. This implies that the on the average there will be $(\sigma+ C + n\gamma)\lambda$ number of requests before the request of the end-host of the *last_NSN* in the distributed queue. The wake-up signal generation rate at the *head_NSN* is α. So it will take $\frac{(\sigma + C + n\gamma)\lambda}{\alpha}$ units of time to process $(\sigma+ C + n\gamma)\lambda$ number of requests. Then it takes another C unit of time to reach the *wake_up* signal from *head_NSN* to the *tail_NSN*. So the total waiting time for the end-host at the *last_NSN* before it acquires the floor will be $\frac{(\sigma + C + n\gamma)\lambda}{\alpha} + C$

+ σ. Using Equation (1), we derive the efficiency of DQDB implementation on participating chain as follows.

$$\eta = \frac{\delta}{\frac{(\sigma + C + n\gamma)\lambda}{\alpha} + C + \sigma + \delta} \quad \ldots \quad \ldots \quad \ldots(4)$$

2.6.3 Results

We compare the efficiencies of different floor control protocols using our analytical model. We set the number of participating NSNs *n* to 10. The average activity time, δ is set to the length of the chain(*C*) which is assigned the value 0.5*s*. The average delay between an end-host and its NSN σ is set to 0.3*s*. γ, the average processing time for control message is set to 0.02*s* [11]. α, the frequency of *wake_up* signal generation rate at the *head_node* for DQDB is set to 1 in each second. Efficiency for each protocol for varying request rate (λ) is calculated using the formula provided in the previous subsection. The results are plotted in Figure 2.11.

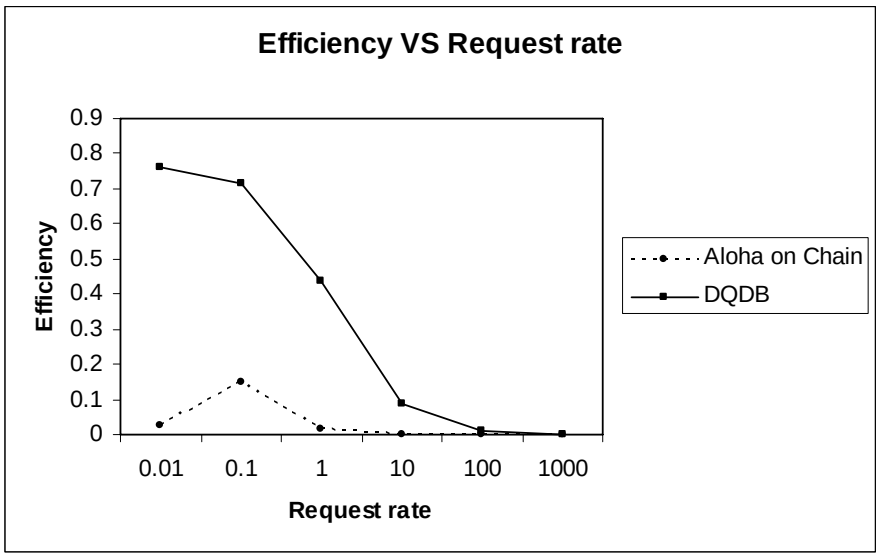

Figure 2.11: Comparing Efficiencies of Different Floor Control Protocols with Analytical Model

From the results, we observe that on average DQDB outperforms ALOHA in terms of efficiency. The efficiency of ALOHA reaches 0 when the request rate is 10 whereas the efficiency of DQDB reaches 0 when the request rate is near 1000.

2.7 Simulation Experiments

We have simulated both ALOHA and DQDB floor control protocols using the latest version of *ns*-2. Simulations were done on a participating chain of 10 NSNs with 2 end-hosts connected to each NSN. In both simulations, we have used *constant bit rate* traffic source with different time intervals and UDP as the transport layer protocol. The

simulation time is set to 2000 seconds. Some parameters used in the simulations are listed in Table 2.2.

Parameter	Value
request packet size	25 bytes
link delay	50 ms
bandwidth	1 Mbps
cell size for DQDB	44 bytes
contention window for ALOHA	20
cell generation rate for DQDB	1.0

Table 2.2: Parameters used in the simulation

We have varied the request rate for the whole network and calculated the total contention time for each protocol. We assume that the floor utilization time is the length of the participating chain. Then we have calculated the efficiency for each protocol using Equation 1. The results of the simulation experiments are plotted in Figure 2.12. Each point in the plots represents average value taken over 30 simulation runs. From the results, we can observe that DQDB performs better than ALOHA in terms of efficiency when the system request rate is high. For ALOHA, when the system request rate is low, the efficiency achieved from simulation experiments is higher than the efficiency calculated from analytical model. This is due to the fact that when the request rate is low, the probability of collision in the system will be low. So the waiting time for the floor grants will be less which increases the efficiency.

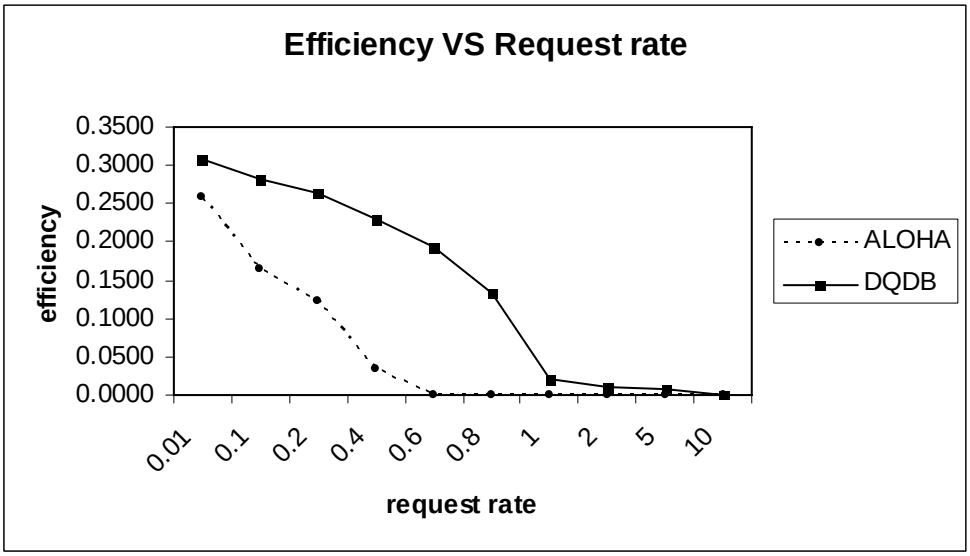

Figure 2.12: Efficiencies VS System Request Rate for Different Floor Control Protocols from Simulation Experiments

2.8 Related Work

Most of the existing floor control mechanisms presented in the literature propose centralized control to coordinate floor among the users. Lennox and Schulzrinne [15] proposed a protocol for Reliable Decentralized Conferencing using a full mesh conferencing system which allows any number of users to participate in a conference without any central point of control. This type protocol is well suited for small impromptu conferences. In this model, every user directly coordinates with other users and all the users have equal rights. The protocol uses several messages for communication between the users and it is assumed that the messages are transmitted

reliably. Though this protocol performs better for small scale conference, its message complexity is very high.

Katrinis et al. [13] proposed a distributed floor control protocol called Activity Sensing Floor Control Protocol (ASFC) using the concept of activity sensing and collision detection from Ethernet protocols. In this protocol, each user monitors the activity on a specific resource and contends to get the floor for the resource only if the it senses that the resource is unoccupied. This sensing is achieved by real-time monitoring of the incoming data traffic towards the resource. An ASFC protocol agent runs at each user end for each resource. In the protocol, a resource is considered to be occupied if the inter-arrival time between two consecutive received data packets at the resource does not exceed a threshold value. The protocol assumes that the maximum inter-departure time between two consecutive data packets transmitted by a floor holder is fixed and globally known. It also assumes that the worst case one-way transit delay experienced by a data packet is fixed. Each user keeps track of number of unsuccessful floor attempts in order to calculate the back-off interval which is an overhead for the end-user. Though the protocol is distributed, it suffers from using fixed parameters which might change from time to time over the network. The protocol also does not ensure causal ordering of data among the end-users.

Qiu et al. [14] proposed a three-channel rotation conference control protocol for an interactive and scalable video-conferencing system. In their protocol, three media channels are used for two interactive speakers. Contention for shared media channels is

resolved using a three channel rotation scheme. Out of three channels, a dedicated channel is used for control traffic. Conference participants can have three roles: current speaker, previous speaker and listener. When a participant wants to become a speaker, he/she sends a request signal to a centralized controller. The controller chooses the speaker depending on the access policy which can be FCFS (First Come First Served) or priority based. Though the protocol makes the collaboration highly interactive, it suffers from the drawbacks of using centralized controller.

The protocol presented by Dommel and Garcia [11] referred as Hierarchical Group Control Protocol (HGCP) uses a logical control tree for coordinating control information among the users. The control tree is derived from the shared acknowledgement tree used in reliable multicast protocol. Nodes in the tree are labeled using a finite alphabet set with the property that label of a node $l(x)$ is the prefix of its children. Three types of control responsibilities are assigned among the nodes in the tree: the control node hosts the floor handler or floor controller which regulates to a resource and updates other nodes in the tree about the status of the resource, the relay node collects control information from its children and forwards them in the tree towards the control node and relays replies to the children, the leaf nodes mark the end of tree branches and communicate with their parent relay nodes. In HGCP, it is assumed that the floor controller and the floor handler are the same node. Every node communicates only with their neighborhood by knowing the direction towards the current floor handler. Floor handler is always the root of the tree and tree virtually rotates with the new floor handler. HGCP works in three phases. In the setup phase,

every node updates its state table by comparing the state tables of its parent and children. The state table contains floor information of about the local and remote resources in the session. Every node advertises its state table to its neighbors in the tree and integrates updates from its neighbors into its state table. In the active phase of the protocol, a node sends REQUEST message to its neighbor which is closer on the path to the floor handler when the node wants to acquire the floor. The direction towards the floor handler is computed using the prefix of the node label entry in the state table. The floor handler orders all the requests based on their priority, queuing, timestamp and reception order. When the floor handler completes its resource access, it sends a GRANT message to its successor using the label information from the state table. When the floor handler receives confirmation from its successor, it relinquishes the floor and multicasts the position of the new floor handler to all the members in the session. Now the nodes contending for the floor sends their REQUESTs to the new floor handler. In the termination phase, floor state information is deleted from the records.

2.9 Summary

Floor control protocols play an important role in collaborative applications on the overlay network. Due to its drawbacks and limitations, centralized floor control solutions are not preferred by the Internet community. In this dissertation, we have proposed to implement two well-known distributed MAC protocols (ALOHA and

DQDB) for floor control on the overlay network. Towards this, we have showed how these protocols can be efficiently implemented using an optimal chain. Among the distributed algorithms, we show that our protocol preserves causal ordering. Using an analytical model, we have derived the efficiency of each protocol. We have also performed simulation experiments for each protocol. Both the analytical and simulation results show that DQDB outperforms ALOHA in terms of efficiency.

Chapter 3

Multicasting with Delay Variation

3.1 Introduction

One approach to ensure minimum delay variation is to buffer the messages at different nodes in the network. This approach can be categorized into three classes: buffering at source node, buffering at intermediate nodes, and buffering at destination nodes. Buffering at source node requires the source node to keep additional information for each destination. The source node will buffer a message for different amount of time for each destination and transmit the message multiple times over the network, and clearly, this is a waste of network bandwidth. Also, buffering at source node defeats the purpose of multicasting which is one of conserving network bandwidth. Buffering at intermediate nodes requires some nodes to be identified as core nodes in the network. Messages are buffered at these core nodes before they are sent to the destinations.

Buffering at destination nodes requires each destination node to buffer the messages before they pass the messages to the application process. In this approach, the source node informs the destination nodes when they can process the received packets. For example, consider the graph in Figure 3.1. In this graph, the value on each curved line specifies the delay of the shortest path from the source to a destination. Suppose the source node S sends packets to destinations D_1, D_2 and D_3. The maximum delay among the paths for destinations D_1, D_2 and D_3 is 5. Now to ensure minimum delay variation among the destinations, the source node S informs destination nodes D_2 and D_3 to process their packets after 5-2 = 3 units of time and 5-1 = 4 units of time, respectively. Node D_1 can process the packet as soon as it receives. This implies that node D_2 and D_3 have to buffer their packets for 3 and 4 units of time respectively. It is easily observed that the amount of buffer space required at each destination node is directly proportional to the maximum delay variation among the destinations. Another major drawback of this scheme is that it relies on the destination nodes for minimizing the delay variation. Such a scheme can be easily compromised by the end users to gain a competitive edge over others in case of online games. To overcome these drawbacks, our approach is to route the packets using longer paths for some destination nodes such that all the destination nodes receive the message almost at the same time. This implies, for node D_2 and D_3 in Figure 3.1, instead of using the shortest paths, we will find longer paths from S to D_2 and D_3 so that they receive the packets at the same time and can process the packet without buffering them.

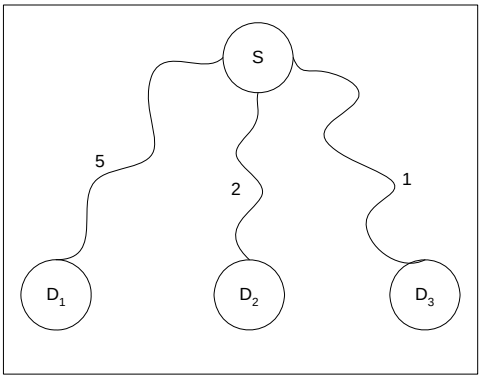

Figure 3.1: A graph with S as source node and D_1, D_2, D_3 as destination nodes

3.2 Formal Definition of the Problem

We use a weighted graph $G = (V, E)$ to denote the overlay network where V denotes the set of end-users (NSNs, in case of proxy based overlay) and E denotes the set of links on the overlay network between the end-users. For each link in E, we define a link-delay function D: $E \rightarrow \Re^+$. Basically the link-delay function associates a delay with each link in the overlay network. In order to do multicasting, a source node $s \in V$ sends multicasting messages to a group of destination nodes $M \subseteq V - s$. The messages are transmitted through a multicasting sub-network $T = (V_T, A_T)$ where T spans the source node s and all the destination nodes in M. The sub-network T may contain nodes other than in M and the source node s. A *path* $P_T(s,v)$ is defined as the path from source s to destination $v \in M$ in T. The total delay for sending message from source s to

destination v along this path will be $\sum_{l \in P_T(s,v)} D(l)$. We now define two important parameters for DVBMN problem.

- Source-destination delay bound, Δ: Parameter Δ refers to the upper bound on the end-to-end delay along any path from the source to a destination node.
- Inter-destination delay variation tolerance, δ: Parameter δ is the maximum allowed difference between the end-to-end delays along the paths from the source to any two destination nodes.

The formal definition of DVBMN problem is stated below (see also [17]).

Given an overlay network, $G = (V, E)$, a source node $s \in V$, a multicast group $M \subseteq V - s$, a link delay function D: $E \rightarrow \Re^+$, a delay bound Δ and a delay variation tolerance δ, find a multicast sub-network $T = (V_T, A_T)$ which spans s and all the nodes in M such that

$$\sum_{l \in P_T(s,v)} D(l) \leq \Delta \qquad \text{for each } v \in M \tag{1}$$

$$\left| \sum_{l \in P_T(s,v)} D(l) - \sum_{l \in P_T(s,u)} D(l) \right| \leq \delta \qquad \forall v,u \in M \tag{2}$$

We define another parameter δ_T, the maximum inter-destination delay variation as follows

$$\delta_T = \max \left| \sum_{l \in P_T(s,v)} D(l) - \sum_{l \in P_T(s,u)} D(l) \right| \text{ over all } u,v \in M \tag{3}$$

The *tightest delay variation* is the one that minimizes δ_T. This notion helps us benchmark whether each heuristic achieves the tightest bound for a delay variation. We have assumed that node latency is zero at each node in the overlay network.

3.3 Literature Review

Multicast Routing with QoS constraints has been extensively studied by the network community due to the popularity of computer collaborative applications which demand different QoS requirements. Many protocols for this problem have been proposed in the literature [36-42]. Salama et al. [43] and Dziong et al. [44] have provided a comparison study of different multicasting protocols. Most of these protocols try to construct a delay-constrained minimum cost tree. Multicast routing with delay and delay variation constraints has been addressed by Rouskas and Baldine [17], Sheu and Chen [19] and Kapoor and Raghavan [18]. In the following subsections, we will discuss the heuristics proposed by Rouskas and Baldine [17], Sheu and Chen [19] and Kapoor and Raghavan [18].

3.3.1 Delay Variation Bound Multicast Algorithm (DVBMA) [17]

The heuristic proposed by Rouskas and Baldine [17] constructs a multicast tree satisfying an end-to-end delay constraint Δ, and delay variation constraint δ. Initially the shortest path tree T_0 from source s to all nodes in multicasting group M is determined using Dijkstra's algorithm [45]. If T_0 does not satisfy the end-to-end delay constraint Δ, then no tree will satisfy it and hence the algorithm terminates, otherwise, if it satisfies both end-to-end delay constraint Δ and delay variation constraint δ, the algorithm outputs T_0 and terminates. It is possible that the delay variation constraint is not satisfied and the algorithm may terminate without finding the tree. The second part of the algorithm in [17], referred as Delay Variation Multicast Algorithm (DVMA), first identifies the farthest destination node w in the shortest path tree T_0. It computes first k-shortest paths from s to w. Then it repeatedly performs the following. It takes a shortest path from s to w, forms an initial tree T_i and tries to attach other destination nodes in M to T_i through the best path (which satisfies the end-to-end delay constraint Δ and has minimum delay variation). This is performed by finding l-shortest paths from each node in T_i to a destination node in M which is not yet connected to T_i. DVMA searches through all possible trees depending on the value of k and l and returns the tree with minimum delay variation. The main advantage of DVMA is that it ensures that the resulting solution is a tree. During our implementation of this algorithm we found a serious drawback in the heuristic in [17]. The order in which the

destination nodes in *M* are added to the initial tree determines the tightness of the delay variation. Of course all possible orderings will make the complexity of the heuristic exponential.

3.3.2 Delay and Delay Variation Constraint Algorithm (DDVCA) [19]

Sheu and Chen [19] heuristic (named DDVCA) is a combination of Core Based Tree (CBT) [46, 47] concept and Dijkstra's shortest path algorithm [45]. In CBT approach, some routers are chosen as Core Routers. In DDVCA, the Core Router is addressed as a central node. The main steps of DDVCA are as follows. First the central node of the network, C_n is determined. In order to find the C_n , minimum delay paths between each destination node and other nodes in the network is calculated using the standard Dijkstra's algorithm [45]. Then for each node in the network, the delay variation between the node and each destination node is determined. The node with minimum delay variation is chosen as C_n. In the next step, the multicasting tree is constructed by connecting the destination nodes and the source node *s* to the central node C_n using the minimum delay paths. It might be possible that the multicasting tree constructed with the chosen central node, violates the end-to-end delay constraint Δ. This implies that the delay of the path from *s* to a destination C_n exceeds Δ. In this case the node with second minimum delay variation is chosen as C_n. The main advantage of DDVCA is

that it is simple and has a time complexity of $O(mn^2)$. But the main disadvantage of this approach is that it does not take into account the delay variation bound δ.

3.3.3 Dynamic Program for Delay Variation Bound (DPDVB) [18]

Kapoor and Raghavan [18] proposed a heuristic (named DPDVB) that tries to minimize the cost of multicasting tree together with the delay and delay variation constraints. The cost can be a measure of residual bandwidth or the amount of buffer space required in the network. This problem can be reduced to DVBMN problem if the costs are set to zero. In this algorithm, it is assumed that the delay and cost values are integrals, implying that the delay and cost values can be scaled to corresponding integer values. The function $g_j(t)$ is defined as the minimum cost path from node 1 to node j with delay exactly equal to t and node 1 as the source. Then the following recurrence is defined

$$g_j(t) = \min \{ g_k(t - t_{kj}) + c_{kj} , \forall k \mid (k,j) \in E, t_{kj} \leq t\}.$$

where c_{kj} and t_{kj} denote the cost and delay of the path from node k to node j, respectively. The algorithm first initializes two variables TC and t to infinity and 1, respectively. TC represents the total cost of all the paths in the solution. Now $g_j(t)$ is computed for all nodes in the network except the source. Then the algorithm checks whether paths exist to all nodes in M in the window of (t-δ). If paths exist to all nodes in M in this window, then the overall cost C for paths to all nodes in M is computed. If

C is same for more than one set of paths, then the set of paths, which have minimum delays for nodes in M, is chosen. If C is less than or equal to TC, then TC is initialized to C. Next t is incremented by 1. The algorithm computes $g_j(t)$ again and the whole process is repeated till t is not equal to Δ. The main advantage of DPDVB is that it achieves the tightest bound in delay variations.

3.4 Motivating Example

Our heuristic works as follows. It first computes the k-shortest paths[5] from source to each of the destinations such that the delay of each shortest path is less than or equal to the delay bound. The value of k is chosen depending upon the size, edge density, and number of destinations. This analysis is shown in section 3.6. We cleverly select a shortest path for each destination from among the k shortest paths available to that destination in such a way that the delay variation is the smallest possible. We will examine this concept with the example below. Consider the overlay network given in Figure 3.2.

[5] K-shortest Path Problem: Given a graph $G = (V, E)$ and two nodes s, $t \in V$, find k shortest paths between s and t in G in order of increasing length.

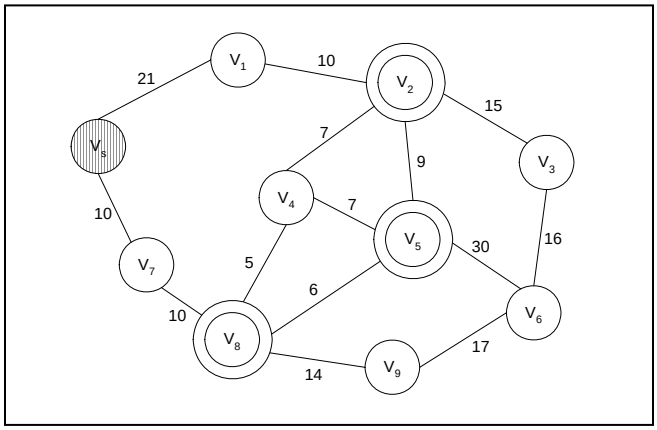

Figure 3.2: Example of a network with link delays. Nodes in double circles are the destination nodes and node V_s is the source node.

In Figure 3.2, we have an overlay network with V_S as source node and V_2, V_5, V_8 as destination nodes. We want to construct a multicasting sub-network with tightest delay variation and end-to-end delay bound of 50. First we find all the paths from V_S to V_2, V_5, V_8 for which the delays are less than or equal to 50. Then we list all the paths with their corresponding delays in the increasing order of the delays as shown in Table 3.1.

Source	Destination	Path	Delay
V_S	V_2	(a) V_S –V_1–V_2	31
		(b) V_S–V_7–V_8–V_4–V_2	32
		(c) V_S–V_7–V_8–V_5–V_2	35
		(d) V_S-V_7-V_8-V_5-V_4-V_2	40
	V_5	(e) V_S–V_7–V_8–V_5	26
		(f) V_S–V_7–V_8–V_4–V_5	32
		(g) V_S–V_1–V_2–V_5	40
		(h) V_S–V_1–V_2–V_4–V_5	45
	V_8	(i) V_S–V_7–V_8	20
		(j) V_S–V_1–V_2–V_4–V_8	43
		(k) V_S–V_1–V_2–V_5–V_8	46

Table 3.1: List of paths from V_S to V_2, V_5, V_8 and their corresponding end-to-end delays

From this list of paths, we choose the delays and their corresponding paths to construct the multicasting sub-network as follows. First we have to choose a path from V_s to each of the destinations from the set of paths given in Table 3.1. For example, let us suppose that we choose paths (a), (e), and (i) (the first shortest paths from source to each of the destinations V_2, V_5, and V_8, respectively), then after merging these paths the

resulting sub-network will be a shortest path tree. If we remove the delay variation constraint from our problem, then the shortest path tree will be the optimal multicasting tree. Note that destination nodes V_2, V_5, and V_8 receive the message from V_s at time units 31, 26, and 20 resulting in a delay variation of 11.

If the paths that are chosen are (d), (g), and (j), the end-to-end delays for each of the destinations V_2, V_5, and V_8 will be 40, 40, and 43, respectively. These delays are within the desired delay bound of 50, but the delay variation is only 3 smaller when compared to 11 in the previous case. Merging these paths we obtain a sub-network as shown in Figure 3.3. Although the sub-network is not a tree, at most one message needs to be sent on each direction of any link. We notice from the multicasting sub-network in Figure 3.3 that destination node V_2 works as relay node for the paths V_S – V_1 – V_2 – V_4– V_8 and V_S – V_1 – V_2 – V_5. That is, the node V_2 will not consume the data while working as a relay node for these paths. It will simply just forward the data along the path. This can be done using source-routing protocols where source specifies the path to be followed by a packet. In Grid Computing [48] Environment, this type of protocol is implemented using software routers.

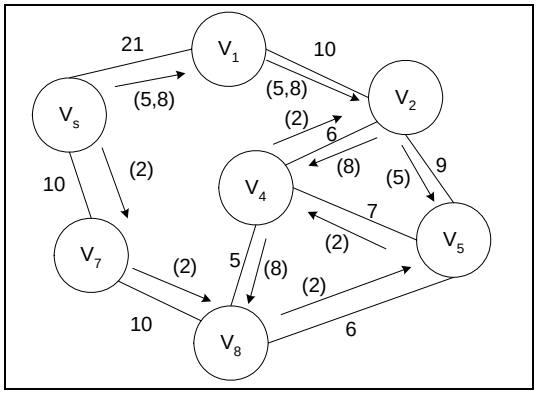

Figure 3.3: The multicasting sub-network with a delay variation of 3 and a delay bound 50 obtained after merging the paths (d), (g), and (j) in Table 3.1. The number in the parenthesis indicates the destination(s) of the packets sent from the source V_s. The directions on the arcs indicate how the messages for each of the destinations travel.

3.5 Our Heuristic: Chains

Given the motivating example in section 3.4, we know that the number of such shortest paths from the source to each destination within the bounded delay may not all be the same (as in the example in the previous section). However, we can assume that the number of shortest paths for any source-destination pair is bounded by an integer value *k*. Now, we can see that after computing all shortest paths for each of the *m* destinations satisfying the maximum delay constraint, we can find a path to each of the destinations that gives the smallest delay variation in a brute-force manner. Clearly, the time-complexity of this approach would be $O(k^m)$. In this section, we present an algorithm that has a time-complexity of $O(m^2k)$ that chooses a path to each of the destinations from a set of *k* shortest paths (to each destination) such that the delay

variation is the smallest. Note that the k-shortest paths from a given source s to all the nodes in a n node network with $|E|$ links can be determined in $O(|E| + nk \log (|E|/n))$ [20, 21]. Given the algorithm in this section, the total time-complexity of the entire algorithm would be $O(|E|+ nk \log (|E|/n) + m^2k)$ for the DVBMN problem.

Suppose source s wants to multicast data to destinations $v_1, v_2, \ldots, v_m$ and let there be k_i different shortest paths for each v_i. Let the delay of these paths be $d_{i1}, d_{i2}, ..., d_{ik_i}$ in the increasing order of the end-to-end delays. Our aim is to find the set of paths P from source s to each destination $v_1, v_2, \ldots, v_m$ such that maximum difference between the delays of these paths in P is a minimum. The problem of finding these paths can be transformed into the following problem.

Consider sets $S_i = \{d_{i1}, d_{i2}, \ldots, d_{ik}\}$, where $1 \leq i \leq m$, where d_{ij}, $1 \leq j \leq k$ are integers with $color(d_{ij}) = i$. Without the loss of generality, assume that the elements in each S_i are non-decreasing. Define $D = \bigcup_{i=1}^{m} \{S_i\}$ and again without loss of generality assume that the elements in D are non-decreasing. The set D can be constructed in $O(mk)$ time using the merge algorithm [45]. We will define $color[i] = j$, for each element i in D if the element i belongs to the set j.

We now construct an array *next*, where $next[i]$ corresponds to the ith element of the set M as defined as follows: The array *next* is of size mk and $next[i] = \min\{j \mid j > i,$

color(j) ≠ color(i), $j \leq mk$}, if there exists such a j, otherwise *next*[i] = -1. For example, consider the following sets:

$S_1 = \{31, 32, 35, 40\}$, $S_2 = \{26, 32, 40, 45\}$, $S_3 = \{20, 43, 46\}$

	1	2	3	4	5	6	7	8	9	10	11
D	20	26	31	32	32	35	40	40	43	45	46
color	3	2	1	1	2	1	2	1	3	2	3
next	2	3	5	5	6	7	8	9	10	11	-1

The *next* array can be computed in $O(mk)$ time by a linear scan of the array *D* starting from the right and moving towards the left. The algorithm to compute the next array is given below:

Algorithm Compute_next_array

Begin

```
t = mk;
c = color[mk];
next[t] = -1;
for ( j = mk -1; j > 0; j --)
{
    if( color[j] ! = c)
    {
        next[j] = t;
        c = color [j];
```

```
            }
            else
            {
                    next[j] = next[t];
            }
            t = j;
    }
End
```

The sequence i, $next(i)$, $next(next(i))$, $next(next(next(i)))$, … is defined as a *candidate chain* starting from the ith element of D. In this case, we say that element i starts a candidate chain. A candidate chain is a *valid chain* if it contains exactly m elements and each element in the candidate chain is of a different color. Each element i that starts a valid chain is called as a *valid element*, otherwise the element will be called as *invalid*. It is easy to observe that all *invalid* elements occur at the rightmost end of the array D. In order to identify the *valid* and *invalid* elements in D, initially we mark all the elements in D as *valid* and scan the array D from right to left. As we move from right to left we keep a count of the number of elements that are of different color (call this variable *count*). We can use a Boolean array of size m initialized to false, to keep track of the different colors processed so far. When we encounter an element i and *count* is not equal to m, we increment the *count* variable by one if the element is not found in the Boolean array (constant time to check if it is present). In this case, the

i^{th} element is marked as *invalid*. When *count* reaches *m*, our algorithm terminates. The complexity of finding all the valid and invalid elements is $O(mk)$. In the above example, elements in position 9, 10, and 11 are invalid.

Also for the example above, the chain {20, 26, 31} is valid, similarly {26, 31, 43}, {31, 32, 43}, {32, 32, 43}, {32, 35, 43}, {35, 40, 43}, and {40, 40, 43} are some of the other valid chains. The *value* of a valid chain is difference between the last and first element of the chain. For example, the value of the valid chain {26, 31, 43} is 43-26 = 17 and similarly the value of valid chain {40, 40, 43} is 40-43 = 3.

Based on the above problem definition and the example, we can say that our next goal would be to determine a valid chain whose value is a minimum among all valid chains. First, we will describe a $O(m^2k^2)$ algorithm Chains to find a valid chain of minimum value and then show how the complexity can be improved to $O(m^2k)$ based on a useful observation. The algorithm Chains assumes that *D*, *color*, and *next* are all available. It also assumes that the valid and invalid elements are marked appropriately.

Algorithm Chains

Begin

BestChainValue = ∞;

BestChainStartPos = 1;

for (int i = 1; i ≤ m·k; i++)

```
{
        //Find the valid chain starting from i
        for (j=1; j ≤ m; j++) flag[j] = false;
        count = 1;
        k = next[i];
        flag[color[k]] = true;
        while ((k ≠ -1) && (count ≠ m))
        //May have to loop through the entire list D
        {
                if !flag[color[k]]
                {
                        flag[color[k]] = true;
                        count = count + 1;
                }
                if (count < m) k = next[k];
        }
        if (count == m)
        //Valid chain found starting from i
        {
                currentChainValue = D[k] – D[i];
                if(currentChainValue< BestChainValue)
                {
                        BestChainValue= currentChainValue;
```

```
                    BestChainStartPos = i;
                }
            }
        }
```

End

The algorithm Chains clearly has a complexity of $O(m^2k^2)$ and using the following Property 1 and Property 2, we will show that the complexity of the algorithm can be improved to $O(m^2k)$. We will define *first*(C) to be an index position in D that is the first element of the chain C; similarly *last*(C) be an index position in D that is the last position of the Chain C.

Property 1: Let C_1 and C_2 be two valid chains with *first*(C_1) < *first*(C_2) < *last*(C_1) and there exists an element at position j in D such that *color*[j] = *color*[*first*(C_1)] and *first*(C_1) < j < *last*(C_1) and $D[j] \in C_2$. The value of chain C_1 is greater than or equal to the value of chain C_2.

Proof: Suppose we have a list D with n elements in non-decreasing order and P, Q, R, S and T denote the color of the elements. We construct two chains C_1 and C_2 from D with C_1 containing elements $P(1)$-$Q(2)$-$R(i_1)$-$S(j_1)$-$T(n)$ and C_2 containing elements $Q(2)$-$P(3)$-$R(i_2)$-$S(j_2)$-$T(n)$. The number inside the parenthesis indicates the index position of the element in D. We can easily observe that C_1 and C_2 satisfy the condition *first*(C_1) < *first*(C_2) < *last* (C_1) and here j = 3 as *color*[3] = *color*[*first*(C_1)] and $D[j]$ is

included in chain C_2. Let us assume that the value of chain C_1 is less than the value of chain C_2. We calculate the value of chain C_1, $value(C_1) = D[n] - D[1]$ and value of chain C_2, $value(C_2) = D[n] - D[2]$. Now $value(C_1) < value(C_2)$ implies that $D[1] > D[2]$. But this is a contradiction as the list D is in non-decreasing order of elements. So value of chain C_1 will be always greater than or equal to the value of chain C_2. ■

Property 2: For any two valid chains C_1 and C_2, if $first(C_1) < first(C_2)$, then $last(C_1) \leq last(C_2)$.

Proof: It can be easily observed that if $first(C_1)$ is less than $first(C_2)$, then chain C_2 will end either along with chain C_1 or after chain C_1. In the first case, $last\ (C_1)$ will be equal to $last\ (C_2)$. The second case will occur when there are some colors which appear between $first(C_1)$ and $first(C_2)$ and then appear again between $last(C_1)$ and $last(C_2)$. In this case, $last\ (C_2)$ will be greater than $last\ (C_1)$. ■

Our improved algorithm works as follows. First we scan all the elements in D (actually the array *color*) from left to right. At each position j we keep an array *count* of size m (the maximum number of colors), where $count[i]$ is the number of elements from positions 1 to j with color i. This process will take $O(m^2k)$ time. We perform a second scan of the array D (along with the *color* array). Before the beginning of the second scan, we create a queue Q that stores nodes that have only one attribute which is the color of the element. The maximum number of nodes in the queue is at most m and no two nodes in the queue have the same color. An array *pointTo* of size m is

created, where *pointTo*[*i*] points to the node with color *i* in the queue, if it exists otherwise it is set to *null*. The node with *color*[1] (the color of the first element of *D*) is created and added to the queue *Q*. The element *pointTo*[*color*[1]] is set to point to this node in the queue *Q*. The scan begins with the second element in *D* (along with array *color*). Conceptually now we will try to find a valid chain that begins with *Q.front*() (the first element of the queue). From Property 2 we will first find the valid chain for this element provided that we do not find another element with the same color as the one in the front of the queue (based on Property 1). We will keep going until we find *m* different colored elements and each time we find a element with a new color we will add that to the queue. If we come across a color that we have already seen then we will remove them from the queue (*O*(1) operation since we have access to it via the *pointTo* array).

If we remove the element from the front of the queue, then based on Property 1, there exists another valid chain that may start with the same of different color. Now rather than rescanning, we will use the information from scan one as follows. Let us say that when the second scan is at position *l* we removed the front of the queue (that is when the *color*[*j*] = *color*[*Q.front*()]). Also, let the new front of queue *Q* be an element that was added to the queue when the scan was at position *p* ($p < l$). Our next goal is to find a valid chain from *p*. To avoid rescanning from position *p*, we can use the count at locations *p* and *l* to determine the number of elements with different colors in locations *p* through *l* and proceed to scan from positions *l* +1 for colors that need to be added to form a valid chain. Because each element of the arrays *D* and each *color* is processed

at most once during the second scan and also that the maximum number of comparisons needed at each position (using the count array) is at most m, the total complexity of the second scan and hence the improved algorithm is $O(m^2k)$.

Based on the above algorithm, we can clearly see that after computing k shortest paths (that satisfies the delay bound) to each of the destinations, a path from source s to each of the destinations can be chosen in $O(m^2k)$ time such that the delay variation is minimized.

Given that the k-shortest path algorithm can be determined in $O(|E| + nk \log (|E|/n))$ [20, 21] the total time complexity of the DVBMN problem is $O(|E| + nk \log (|E|/n) + m^2k)$.

3.6 Performance Evaluation

For evaluation purposes, we implemented DVMA, DDVCA, DPDVB and our heuristic Chains and compared each of them in terms of the tightest delay variation and execution time. We set the end-to-end delay bound $\Delta = 0.05$s and the delay variation bound $\delta = 0$ (this value of δ will force each heuristic to return the solution with the tightest δ_T). We have run the heuristics on randomly generated graphs constructed using Georgia Tech Internetwork Topology Models (GT-ITM) [49, 50]. The nodes in the graph are placed in a grid of dimension 4900×4900 km to resemble the whole

network spreading all over the United States and delay for each link is set to the propagation delay of electrical signal along that link. The average node degree for each graph is kept in the range of 3.5 and 4. Each heuristic has been tested on various graphs with number of nodes varying from 20 to 100 and percentage of nodes in the multicast group varying from 5% to 15%. The results are plotted in Figures 3.4 through 3.6. Each point in the plots represents average value taken over 30 graphs. The experiments are done on Sun UltraSPARC-IIi Workstations (360 MHz of clock speed and 128 MB RAM) running Solaris 8.

From Figures 3.4 - 3.6, we observe that Chains along with DPDVB achieves the tightest delay variation for all the cases. We also notice that Chains outperforms DVMA and DPDVB for all cases in terms of execution time. Only DDVCA shows better performance than Chains in terms of execution time. But we have already observed that DDVCA does not achieve the tightest delay variation. Table 3.2 shows the numerical values of execution time of DVMA, DPDVB, DDVCA and Chains for various numbers of nodes. Table 3.3 shows how the execution time increases when the percentage of nodes in the multicast group increases for DVMA, DPDVB, DDVCA and Chains. We observe that Chains outperforms DVMA and DPDVB in terms of execution time even with higher number of nodes in the multicast group. Since DPDVB along with only Chain achieves the tightest delay variation and from Table 3.2 we observe that the rate of increase in the execution time of Chain is more than that of DPDVB, we have performed additional experiments with Chain and DPDVB for more number of nodes. From Table 3.4, we can easily observe that the execution time of

DPDVB is always more than that of Chains even with higher number of nodes. We also observe that the rate of increase in the execution time of Chains decreases for higher number of nodes. This is due to the fact that execution time of Chain depends largely on the value of k. As the value of n (number of nodes) increases, Chain requires larger value of k to find out the multicasting sub-network with minimum δ. But the rate of increase of k goes down as the value of n increases. This implies that for higher values of n and fixed Δ, a larger value of k might not be required by Chain to obtain a multicasting sub-network with minimum δ. Next we compared the execution time of DPDVB and Chains for different values of Δ, the end-to-end delay bound. The results are plotted in Figure 3.7. From Figure 3.7, we observe that the execution time of DPDVB increases with Δ, whereas the execution time of Chains remains almost constant.

The asymptotic complexity of our algorithm is dominated by finding the k-shortest paths from source to all destinations such that the delay for these paths are less than or equal to the delay bound Δ. If Δ is large, then the value for k may increase. However, our implementations show that finding k-shortest paths in a large network with the algorithm proposed in [20] is quite efficient. For our experiments, we have generated random graphs consisting of nodes 50 and 100 with a varying edge probability of 0.2, 0.4, 0.6 and 0.8 using the model in [49, 50]. Then we have used the recursive enumeration code for finding k-shortest paths provided at [21]. We have run our simulation for all pairs k-shortest paths with different values of k for each generated

random graph. We also calculated the elapsed time for each run and plotted the results in Figure 3.8. From the plots, we can observe that for a graph with 50 nodes and edge probability 0.8 (which means a dense graph), the time to calculate 10 shortest paths for all pairs is 1.37 sec. When the number of nodes increases to 100, this value becomes 18 sec.

Now we need to determine how k is related to the end-to-end delay bound, Δ and the density of the network. We have done an experiment with Chains for various values of average node degree of the network and the delay bound Δ and calculated the value of k required for achieving the tightest delay variation. We have varied the average node degree from 4 to 8. In order to vary the delay bound Δ, first we calculated the delay of the farthest destination in the multicast group and then set Δ in the range of 100 % to 350 % of the delay of the farthest destination. From Figure 3.9, we observe that as the average node degree of the network increases, which implies the graph becomes dense; we need a higher value of k to achieve the tightest delay variation. This is obvious from the fact that as the graph becomes dense, we have more paths from the source to the destinations. We also observe that as Δ increases, the value of k increases since the search space for Chains increases.

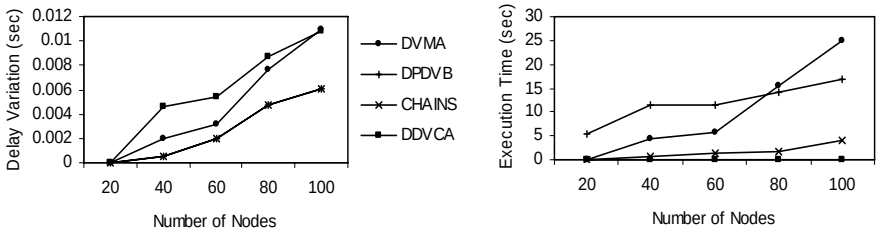

Figure 3.4: Delay variation graph and execution time graph for 5% of the nodes as destinations

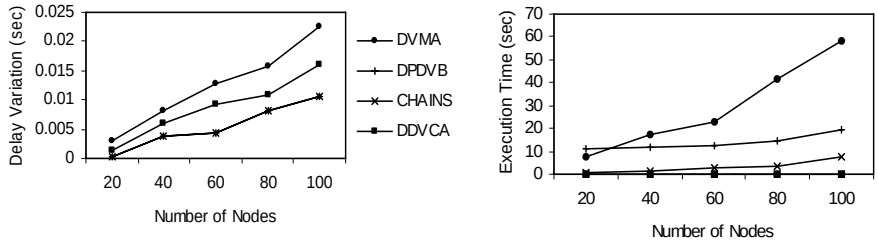

Figure 3.5: Delay variation graph and execution time graph for 10% of the nodes as destinations

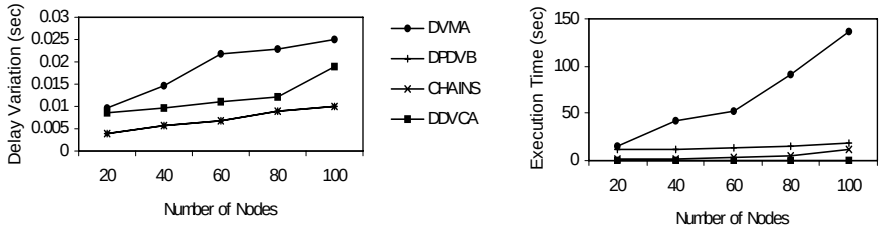

Figure 3.6: Delay variation graph and execution time graph for 15% of the nodes as destinations

Number of Nodes	DVMA (sec)	DPDVB (sec)	DDVCA (sec)	Chains (sec)
20	7.53	11.328	0.0	0.82
40	17.14	11.49	0.01	1.62
60	23.2	12.74	0.02	2.43
80	41.45	14.76	0.065	3.17
100	58.407	19.41	0.12	7.63

Table 3.2: Execution Time vs Number of Nodes with 10% of the nodes in the Multicast Group

% Of Nodes in the Multicast Group	DVMA (sec)	DPDVB (sec)	DDVCA (sec)	Chains (sec)
5	15.642	14.16	0.052	1.55
10	41.45	14.76	0.065	3.17
15	90.49	15.4	0.095	4.664

Table 3.3: Execution time vs % of nodes in the Multicast Group (Total Nodes = 80)

Number of Nodes	DPDVB (sec)	Chains (sec)
20	11.328	0.82
40	11.49	1.62
60	12.74	2.43
80	14.76	3.17
100	19.41	7.63
120	37.05	16.42
140	77.66	19.54
160	119.39	23.42
180	183.05	27.27
200	235.70	38.13

Table 3.4: Comparison of Execution Times of DPDVB and Chains with 10% of the nodes in the Multicast Group

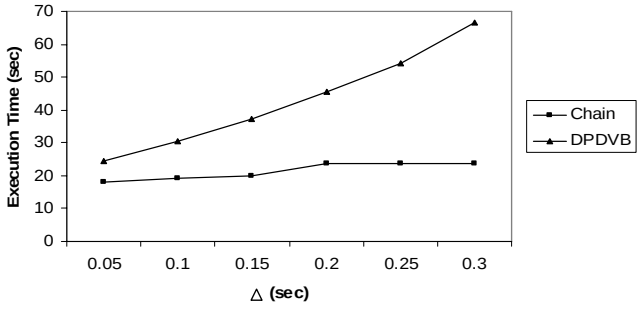

Figure 3.7: Comparison of Execution Times of DPDVB and Chains for different values of Δ (End-to-End Delay bound) with number of nodes 100

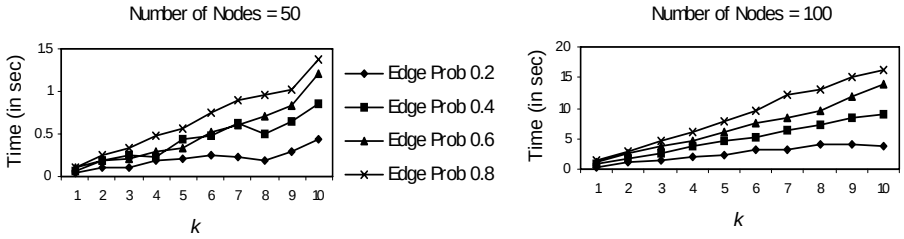

Figure 3.8: Execution time for finding *k*-shortest paths when the number of nodes is 50 and 100

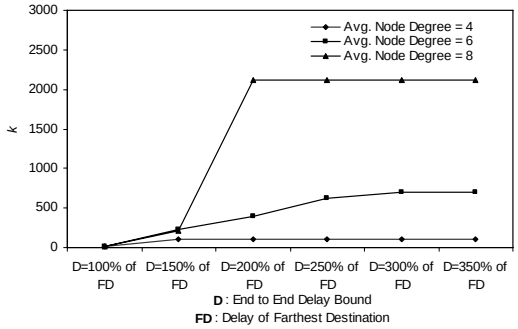

Figure 3.9: *k* vs End-to-End Delay bound for networks with average node degree

3.7 Dynamic Reorganization of Multicasting Sub-Network

In multicasting applications, it is observed that some nodes in the multicast group may leave the session of multicasting after a certain period or some new nodes may join the multicasting group. We need to update the multicasting sub-network to include the joining node and exclude the leaving node in such way that delay along the path from source to each multicasting node (including the new one) is bounded by Δ and the delay variation among the multicasting nodes is a minimum. The authors in [17] discuss various cases for join and leave operation with their heuristic.

3.7.1 Leave Operation

When a node wants to leave the multicasting session, it issues a *leave* request. The leaving node can be either a leaf node or a non-leaf node of the multicasting sub-network.

- If the leaving node is a leaf node, authors in [17] have suggested pruning the tree to exclude the leaf node. But this strategy may not work when we want to achieve the tightest delay variation. In this situation, DVMA has to rework the whole process with updated multicasting group. In Chains, we will remove all

the delays of the leaving node from the sorted list *D* and from *color*, form the chains again and return the chain with minimum delay variation.

- If the leaving node is not a leaf node, the solution of DVMA is to make the node work as a relay node and use the same tree. This solution also may not yield the tightest delay variation and again the DVMA heuristic has to repeat the whole process. For Chains, the solution will be the same as the previous one.

3.7.2 Join Operation

When a node wants to join the multicasting group, it issues a *join* request. The joining node may or may not be part of the existing multicasting sub-network.

- If the joining node is a part of the sub-network, then DVMA uses the same tree with making the joining node work as both a relay node and a multicasting node. If the tightest delay variation is required, DVMA has to rework the whole process with the updated multicasting group. In Chains, we will find *k*-shortest paths from the source to the joining node such that delay of these paths are less than or equal to Δ. Then we will merge this sorted list of delays of the joining node with the existing sorted list of delays of other multicasting nodes. We will form chains and return the chain with minimum delay variation.
- If the joining node is not a part of the sub-network, DVMA finds *l*-shortest paths from each node in the sub-network to the joining node and returns the

best path for constructing the new tree. For Chains, the solution will be the same as in the previous case.

The heuristic DPDVB [18] is unsuited for join and leave operations, as the entire heuristic has to be rerun.

3.8 Summary

In this research, we have considered the problem of determining a multicasting sub-network with end-to-end delay bound and delay variation bounded for collaborative applications on overlay network. We have discussed three well-known heuristics from the literature and exposed their limitations. Then we have presented our heuristic Chains, which achieves the tightest delay variation for a given delay bound. At the initial phase of our heuristic, we have used *k*-shortest path technique proposed by Victor and Andres [20] to find all paths for each destinations for which the delays are less than or equal to the delay bound. Then using these delays, we have determined the delay chain, which gives the minimum delay variation and constructed the multicasting sub-network by retrieving the paths from the delays. We have implemented all the heuristics and observed Chains outperforms DPDVB and DVMA in terms of execution time. The Chains heuristic also achieves the tightest delay variation bound along with DPDVB. We have also presented results to show that finding *k*-shortest paths for all destinations is not a bottleneck in our solution. We have also observed that Chains

require higher values of k to achieve the tightest delay variation when the graph becomes dense or when the end-to-end delay bound increases. For dynamic reorganization of multicasting sub-network with the tightest delay variation and bounded delay, we notice that solution with Chains is more efficient than that of DVMA in terms of time-complexity.

Chapter 4

Duplicating Delays in Overlay Multicasting

4.1 Introduction

Two basic communication models are used to characterize multicast operation on a network. In the first model, known as *telephone* model, a node may send a message to at most one other node in each round. In this model, both the sender and the receiver are busy during the whole sending process. The second model which is a realistic model is known as *postal* model. In the postal model, a sender may send another message before the current message is completely received by the receiver. Bar-Noy et. al. [51] first introduced the heterogeneous postal delay model in the context of network multicasting. In their model, they consider link delays and switching time delay at each

node, and further assume that the time interval between two successive message sends is equal to the switching time. Assume node u has two children v_1 and v_2 and switching time at node u is s_u. Node u sends message to v_1 at time $t = 0$ and the message arrives at v_1 at time $t_1 = \lambda_{uv_1}$, where λ_{uv_1} is the delay of the link (u, v_1). Now u can send a message to v_2 at time $t' = s_u$. The message arrives at v_2 at time $t_2 = s_u + \lambda_{uv_2}$ where λ_{uv_2} is the delay of the link (u, v_2). In this model, the authors assumed that s_u is smaller than λ_{uv_1} and λ_{uv_2}. Eli et. al. [52] modified the heterogeneous postal model and proposed the generalized heterogeneous postal (GHP) model where $t_1 = s_u + \lambda_{uv_1}$ and $t_2 = 2*s_u + \lambda_{uv_2}$. Both Bar-Noy et. al. [51] and Eli et. al. [52] provide approximation algorithms for minimum-delay multicast scheme for a general graph as input.

In this research, we propose a model where node u has different switching time for each child node v (represented using a delay vector) and the message arrival time at each child v depends on the order in which u chooses to send the messages. This model captures the heterogeneous nature of communication links and node hardware on the *overlay network* [1-5]. For example consider an overlay network with hosts H_1, H_2, and H_3. The host H_1 is connected to host H_2 through Fiber Optics backbone network and host H_1 is connected to host H_3 through Ethernet. When H_1 wants to multicast message to H_2 and H_3, the switching time at H_1 for H_2 and H_3 will not be the same. Different switching times for different children induces the notion of ordering at the sending

node and the delay of a multicasting scheme depends on the ordering at each sending node.

Suppose a source NSN S wants to send the same message to a group of destination NSNs D. The general strategy of accomplishing this task is to construct an overlay multicasting (rooted) tree T such that T contains S as the root and NSNs in D as the leaf nodes. The non-leaf nodes and the links of the tree are other NSNs and links of the overlay network, respectively. If the root S has k children in the tree T, then S duplicates the message k times and sends a single message to each of its children. The non-leaf nodes receiving the message, performs the same function as the root and this process is repeated until the leaf nodes (the multicast group) receives the message.

We consider a proxy based overlay network represented by a graph $G = (V, E)$ with n NSNs and m links, where V and E are a set of NSNs (hosts, routers) and a set of links, respectively. Each link $e(i, j) \in E$ is associated with *delay* $d(e) > 0$. Consider a *simple directed path* (simply referred as a path) P from i_0 to i_k (denoted $i_0 \sim i_k$) given by $(i_0, i_1), (i_1, i_2), \ldots, (i_{k-1}, i_k)$, where $(i_j, i_{j+1}) \in E$, for $j = 0, 1, \ldots, (k-1)$, and all $i_0, i_1, i_2, \ldots, i_k$ are distinct. The *path-delay* of P is given by $d(P) = \sum_{j=0}^{k-1} d(e_j)$, where $e_j = (i_j, i_{j+1})$. Let S be a node in the network, called the source node, and $D = d_1, d_2, \ldots, d_k$, where $k \leq n-1$ be the set of destination nodes. The *tree-delay* of the tree T is given by $d(T) = \underset{P_i}{Maximum}\ d(P_i)$, for all $1 \leq i \leq k$, where P_i is path from S to $d_i \in D$. The objective of

multicasting algorithms known in the literature is to construct the tree T that has the minimum $d(T)$.

Banikazemi [53] discusses algorithms such as Spanning Trees, Reverse Path Broadcasting (RPB), Truncated Reverse Path Broadcasting (TRPB), Reverse Path Multicasting (RPM), Steiner Trees (ST), and Core Based Trees (CBT) for building a multicasting tree. The algorithm (TM) due to Takahashi and Matsuyama [54] is a shortest-path based algorithm and was further studied and generalized by Ramanathan [55]. The algorithm (KMB) by Kou, Markowsky, and Berman [56] is a minimum spanning tree based algorithm. None of the algorithms presented consider duplicating delays in multicasting.

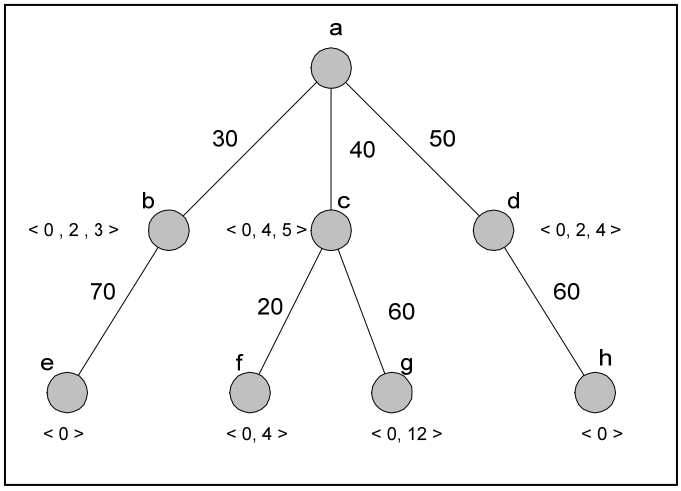

Figure 4.1: Multicasting tree with link delays and duplicating delay vectors at each node.

Figure 4.1 show a multicasting tree with root node ‘a’ and the duplicating delay vectors at each node. The values on the links are the link delays. The leaf nodes of the tree are the nodes in the destination. If we consider only the link delays, the delay of this multicasting tree is 110 as it is the maximum of the delays of all the paths a ~ e, a ~ f, a ~ g and a ~ h. Now, the ordering of packet sends at each non-leaf node will cause additional delay in multicasting as shown in the duplicating delay vectors at each node. As seen in Figure 4.1, when ‘a’ is sending packets to ‘b’, ‘c’ and ‘d’ in the order of ‘b, c, d’, nodes ‘c’ and ‘d’ will incur additional delay due to processing of packet for ‘b’ before them. The duplicating delay vector at node ‘b’ in Figure 4.1 <0, 2, 3> means that if ‘a’ sends packet first to ‘b’, the duplicating delay at ‘b’ is 0. If ‘a’ sends packet second to ‘b’, the duplicating delay at ‘b’ is 2 and if ‘a’ sends the packet third to ‘b’, the duplicating delay at ‘b’ is 3. If the orderings of packet sends at nodes ‘a’ and ‘c’ are ‘b, c, d’ and ‘f, g’, respectively, the delay of the multicasting tree becomes 116 (this is the delay of path a ~ g which is 40 + 4 + 60 + 12 = 116).

Given a multicast tree with link delays and duplicating delay vectors, our goal is to determine the order in which the data packets have to be sent to each of the children in the multicast tree in such a way that the delay of the multicast tree is a minimum.

4.2 Formal Definition of the Problem

4.2.1 Definitions

We define a vector T as an ordered collection of elements, namely, $<v_1,v_2, ...,v_k>$. In other words, given a set $S = \{s_1,s_2,..., s_k\}$, a vectorization of S would involve some specific ordering of elements of S resulting from a bijection $f: S \rightarrow \{1,...,|S|\}$, where $|S|$ is the cardinality of S. Given a vector T, we might use T, ambiguously, as an (unordered) set when ordering information is not relevant to the discussion. For a vector $T = <v_1,v_2, ...,v_k>$, we define a bijective mapping function $\sigma : \{v_1,v_2, ...,v_k\} \rightarrow \{1,2,...,k\}$ such that $\sigma(v_j) = j$, $1 \leq j \leq k$. Let $C = \{C_i\}$, $1 \leq i \leq k$, be a collection of vectors each having the same cardinality k. This implies, each C_i would look like $C_i = < v_1^i, v_2^i, ..., v_k^i >$, $1 \leq i \leq k$. A *feasible vector of representatives* of C is a vector $<v_1,v_2, ...,v_k>$ such that $v_i \in C_i$, and $\sigma(v_i) \neq \sigma(v_j)$, $i \neq j$, $1 \leq i,j \leq k$.

Example: Let $C_1 = <0, 2, 1>$, $C_2 = <2, 0, 3>$, and $C_3 = <1, 2, 3>$. A feasible vector of representatives for the collection of sets $\{C_1, C_2, C_3\}$ is $<2, 3, 1>$, whereas $<2, 0, 1>$ is not.

The following observations are easy to derive.

Proposition 1: Given a collection $C = \{C_i\}$, $1 \leq i \leq k$, of vectors, a feasible vector of representatives of C always exists.■

Proposition 2: Given a collection $C = \{C_i\}$, $1 \leq i \leq k$, of vectors, there exists $k!$ possible feasible vectors of representatives. Let $\mathfrak{I}(C)$ denote the collection of all possible feasible vectors of representatives.■

We will denote the set of non-negative real numbers by the notation $\mathfrak{R}^+$. The cartesian product of the set of non-negative real numbers k times will be denoted by $\mathfrak{R}^+_k$, i.e., $\mathfrak{R}^+_k = \mathfrak{R}^+ \times \mathfrak{R}^+ \times \ ... \times \mathfrak{R}^+$ (k times). Let $T = (V, E)$ be a tree with root r that represents the multicasting network of nodes. Let $S(v)$ denotes the number of siblings of a node v of T, including itself. Trivially, $S(r) = 1$, for the root node. We can model the problem of multicasting as follows based on assigning labels or weights to edges of the multicasting tree. For each node $v \neq r$, there is a vector called the *duplicating delay vector* $D(v) = < t_1, t_2, ..., t_k >$, where $k = S(v)$ and $1 \leq i \leq k$ and $t_i \in \mathfrak{R}^+$. The t_i's are called duplicating time delays. We know that $t_1 = 0$ for all non-root nodes in the tree. However, this fact is not material to the algorithm discussed here. Given a non-leaf node v, let $v_1, v_2, \ ... \ , v_k$ be the children of v. Let us denote the edge set $\{(v, v_1), (v, v_2), ... \ , (v, v_k)\}$ by $E(v)$. We define a *feasible duplicating delay vector* for the edge set $E(v)$ as

$$P_v : E(v) \rightarrow \mathfrak{R}^+_k \text{ such that } P_v = < p_1, p_2, ..., p_k > \in \mathfrak{I}(\{D(v_i) : 1 \leq \ i \leq k\})$$

where $v_1, v_2, ..., v_k$ are the children of v. A feasible duplicating delay vector P_v induces a natural labeling function l_v: $E(v) \rightarrow \mathfrak{R}^+$, where $l((v, v_i)) = p_i$,, $1 \leq i \leq k$. Intuitively, a feasible duplicating delay vector assigns a "label" or a "weight" p_i to each edge (v, v_i) where $< p_1, p_2, ... \ , \ p_k >$ is a feasible vector of representatives for the collection $\{D(v_i)\}$,

$1 \le i \le k$. We call the functions f_v, "feasible duplicating delay functions". Given a multicast tree T rooted at r, and delay vectors $D(v)$ for each non-root node v, we can extend the feasible duplicating delay functions f_v to the whole tree T as follows: A *feasible multicast tree assignment* $f_T : E(G) \rightarrow \mathfrak{R}^+$ such that $f_T((u, v)) = f_u((u, v))$, where $(u, v) \in E$. Essentially, a feasible multicast tree assignment assigns a label or a weight to each edge of the tree so that the collection of weights on an edge set $E(v)$ forms a duplicating delay vector.

Given a leaf node v, we know that there exists a unique path $P = (v_1 = r, v_2, \ldots, v_k = v)$ from r to v. Let f_T be a feasible multicast tree assignment. We define a *path delay* $PD(v)$ as $PD(v) = \sum_{i=0}^{k} f_T(v_i, v_{i+1})$. Given f_T, we denote the *maximum delay* of f_T by $PD_{max}(f_T) = max\{PD(v)$: v is a leaf node of $T\}$. We define an *optimal multicasting tree assignment* as a feasible multicast tree assignment f_T^{OPT} such that $PD_{max}(f_T^{OPT}) = min\ \{PD_{max}(f_T)$: for all feasible multicast tree assignments f_T for $T\}$. We will call $PD_{max}(f_T^{OPT})$ or simply $PD_{OPT}(T)$, the *optimal multicasting duplicating delay* for T. The problem is to compute both f_T^{OPT} and $PD_{OPT}(T)$ in an efficient manner. To solve this problem, we consider the following min-max matching problem and establish a relationship.

4.2.2 Min-Max Matching Problem on Weighted, Bipartite Graphs

Let $G = (S, D, E)$ be a weighted, complete bipartite graph where S and D are the vertex set partitions and E the edge set of G. Furthermore, let us assume that $|S| = |D|$, and that the weights are from $\Re^+$. A *perfect matching* for G is a set of edges M of G such that no two edges of M are incident on a common vertex of G and M has maximum cardinality with this property. For G, trivially, a perfect matching having $|S|$ edges exists. The problems of computing a matching of maximum cardinality and a perfect matching are well studied in the literature [57]. We define *heavy weight* of a perfect matching M for G as $h(M) = max$ {weight of edge e: $e \in M$}. A *min-max* matching of G is a perfect matching N of G such that $h(N) = min$ {$h(M)$: M is a perfect matching of G}. The problem of min-max matching and its dual the max-min matching are problems of independent interest and arise in many scheduling applications. The following lemmas address the complexity of computing a min-max matching for a complete, weighted bipartite graph G.

Lemma 1: The sequential time-complexity for obtaining a min-max matching of a weighted, complete bipartite graph is the same as finding the maximum cardinality matching of a bipartite graph [58][59].■

Lemma 2: Given a complete, weighted bipartite graph, a maximum weighted matching can be determined in $O(m\sqrt{n})$ time [57].■

The above result of [57] was improved by [60] in 1995 to $O\left(\frac{m\sqrt{n}}{k(m,n)}\right)$ where $k(x, y)$ $= \frac{\log x}{\log \frac{x^2}{y}}$

4.2.3 A Special Case of the Multicasting Tree Problem

Let us consider a degenerate case *fan* of the multicast tree. A *fan* $T = (V, E)$ is a multicast tree with $k+1$ nodes, where k of the $(k+1)$ nodes are leaves attached directly to the root node. To be more descriptive, let us also say that the leaf nodes are v_1, v_2, ... , v_k attached to the root v. Let $D_i = < t_1^i, t_2^i, ..., t_k^i >$, $1 \le i \le k$, be the duplicating delay vector for node v_i. We construct a weighted, complete bipartite graph $G = (S, D, F)$ from T as follows. We let $S = \{ v_1, v_2, \ ... \ , v_k\}$, $D = \{1, 2, ..., k\}$, and the edge set $F = \{(v_i , j)$: $1 \le j \le k$, $1 \le i \le k\}$. In other words, each vertex of S is connected to all of the vertices of D. The weight of an edge $e = (v_i , j) \in F$ is given by $w((v_i , j)) = t_j^i$, $1 \le i , j \le k$.

It is fairly straightforward to see that a feasible duplicating delay vector of T is a vectorized representation of the set of weights in a weighted, perfect matching M of

G where the ordering is from 1 through k. Secondly, because T is a fan, $f_r(T)$ is the same as f_T, where r is the root node of T and for all multicasting tree assignments of f_T. Thirdly, the path delay $PD(v_i) = f_T(r, v_i)$ for each leaf node v_i. Hence given a multicasting tree assignment f_T, the maximum delay $PD_{max}(f_T)$ is the heavy weight of the corresponding weighted, perfect matching on G. In the same vein, it is easy to see that an optimal multicasting tree assignment for T can be obtained by finding a min-max matching for the transformed graph G. Finally, the construction of G from T can be done in time $O(n^2)$, where n is the number of nodes in the fan. The number of edges in the bipartite graph is n^2. Based on the above remarks and Lemmas 1 and 2, the following lemma can be obtained.

Lemma 3: Given a multicasting fan T, a special case of a tree, an optimal multicasting tree assignment for T and the corresponding optimal multicasting duplicating delay can be found in $O(n^{5/2})$ time, where n is the number of nodes in T. ■

4.2.4 Hook-up Fans

We will use the notation $F(p)$ for a fan with p leaves, having $(p + 1)$ nodes including the root. Given a collection of vertex-disjoint fans $F(p_1)$, $F(p_2)$,……, $F(p_j)$ with roots $r_1, r_2, \ldots, r_j$ respectively, a hook-up fan is defined as the composition of the collection of fans $F(p_i)$ $1 \leq i \leq j$, such that the hook-up fan is a tree $T = (V, E)$ satisfying the following properties.

1. $V(T) = \bigcup_{i=1}^{j} V(F(p_i)) \cup \{r\}$ Where $V(.)$ denotes the vertex set and r the root of T.

2. The edge set of T, $E(T) = \bigcup_{i=1}^{j} E(F(p_i)) \cup \{(r, r_i) : 1 \le i \le j\}$

Diagrammatically, the hook-up fans obtained by the composition operation looks as shown in Figure 4.2:

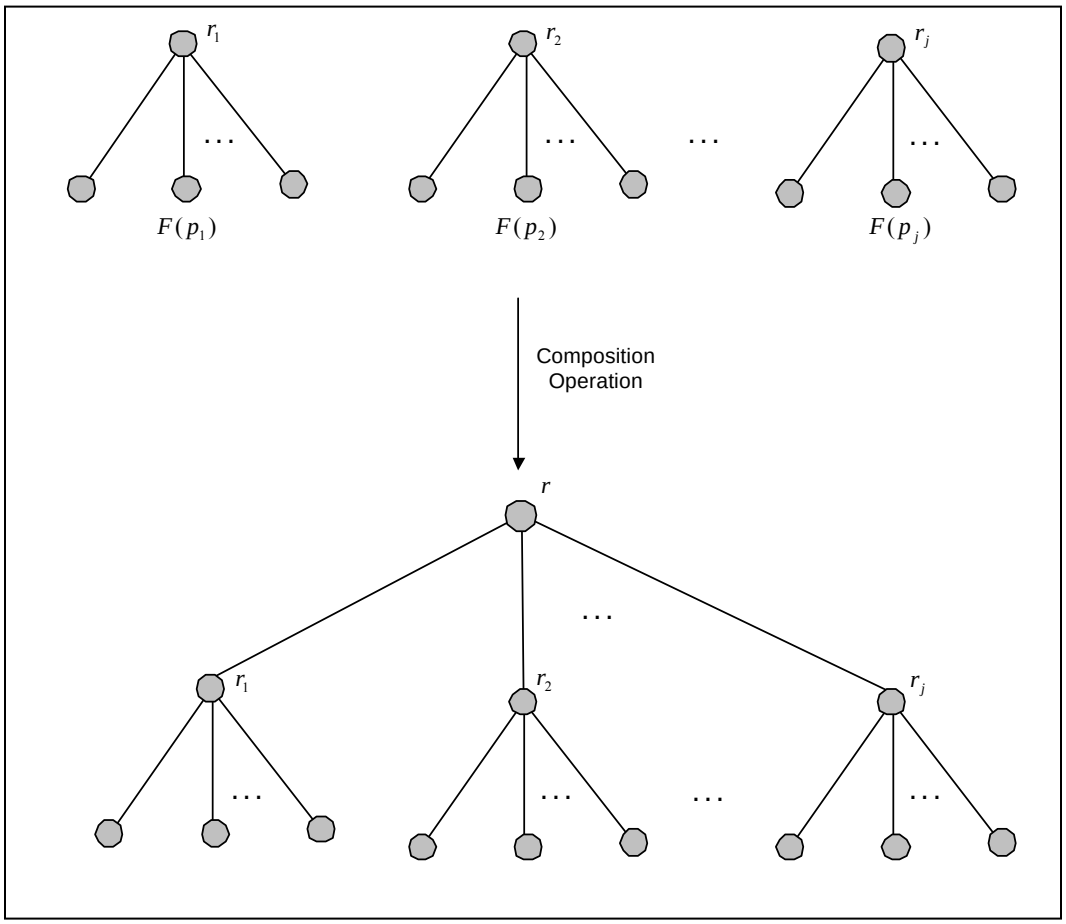

Figure 4.2: Hook-up fan

We denote a hook-up fan by $C(F(p_1), F(p_2), \ldots \ldots, F(p_j))$.

4.2.5 Optimal Multicasting Tree Assignment for a Hook-up Fan

We know from the previous section how to compute an optimal multicasting tree assignment for a fan. In this section, we will show a method to obtain an optimal multicasting tree assignment for a hook-up fan. Consider a hook-up fan H with duplicating delay vector as shown in Figure 4.3:

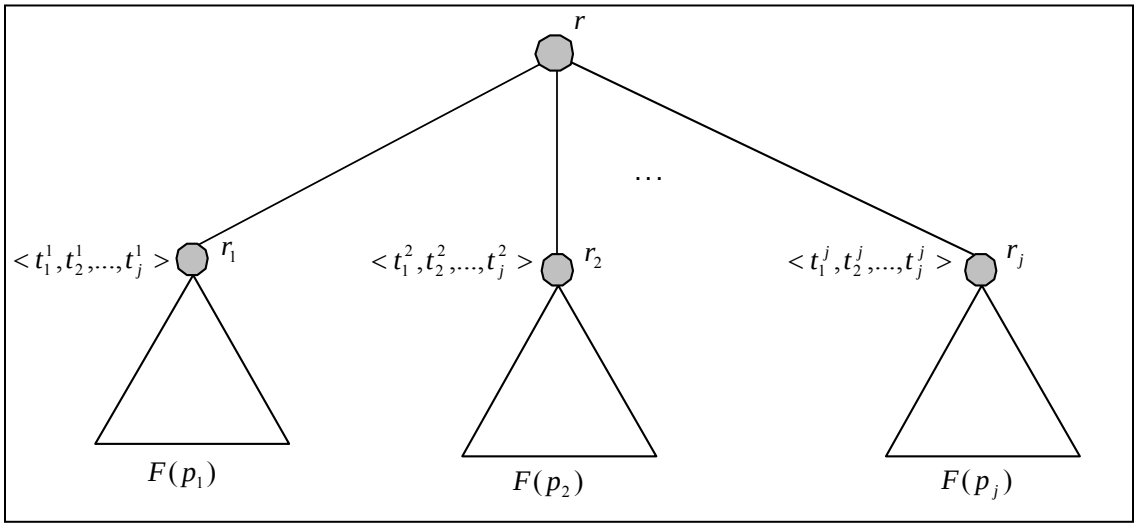

Figure 4.3: Hook-up fan H with duplicating delay vectors

The duplicating delay vectors at nodes r_i are indicated in the Figure 4.3 as

$$D(r_i) = < t_1^i, t_2^i, ..., t_j^i > \qquad \text{for } 1 \le i \le j$$

Let $m_1, m_2, \ldots \ldots, m_j$ be the optimal multicasting duplicating delays for fans $F(p_1)$, $F(p_2)$, …. …., $F(p_j)$ respectively. We know that these can be obtained by using Lemma 3. Let $f_{F(P_i)}$, $1 \le i \le j$ be the corresponding optimal multicasting fan

assignments. We transform the hook-up fan to a fan $F(j)$ as shown in Figure 4.4 along with new duplicating delay vectors.

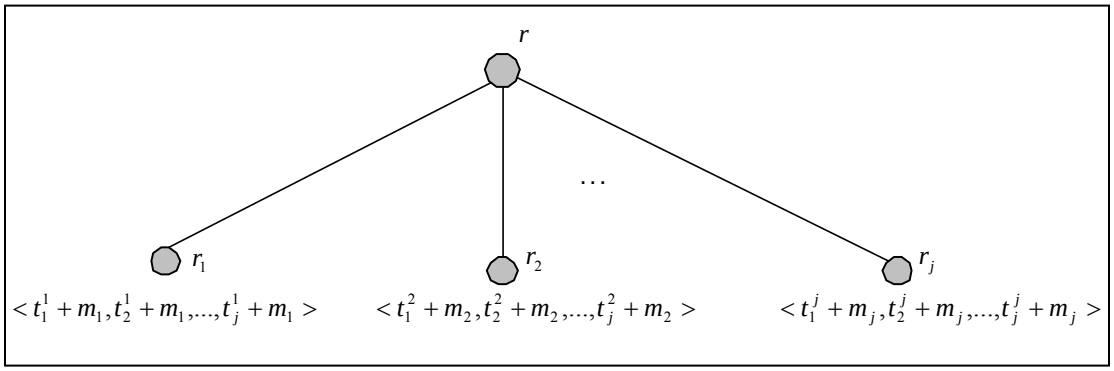

Figure 4.4: Fan $F(j)$ with new duplicating delay vectors

The duplicating delay vectors for the fan in Figure 4.4 are

$$D(r_i) = < t_1^i + m_i, t_2^i + m_i, ..., t_j^i + m_i > \text{ for } 1 \le i \le j$$

We now compute an optimal multicasting tree assignment $f_{F(j)}{}^{OPT}$ for fan $F(j)$ and the corresponding optimal multicasting delay $PD_{OPT}(F(j))$.

Let $f_{F(j)}{}^{OPT} = < l_1, l_2, \ldots \ldots, l_j >$.

We know that each l_i is of the form $v_{r_i}^i + m_i$, $1 \le i \le j$ where $v_{r_i}^i \in D(r_i)$ of H. Secondly, $< v_{r_1}^1, v_{r_2}^2, ..., v_{r_j}^j >$ is a feasible duplicating delay vector for edge set $E(r)$ in H. Based on this, we will re-work the solution obtained on $F(j)$ as a solution for the original hook-up fan H as indicated in Figure 4.5. Let $f_r(r_i) = v_{r_i}^i$, $1 \le i \le j$.

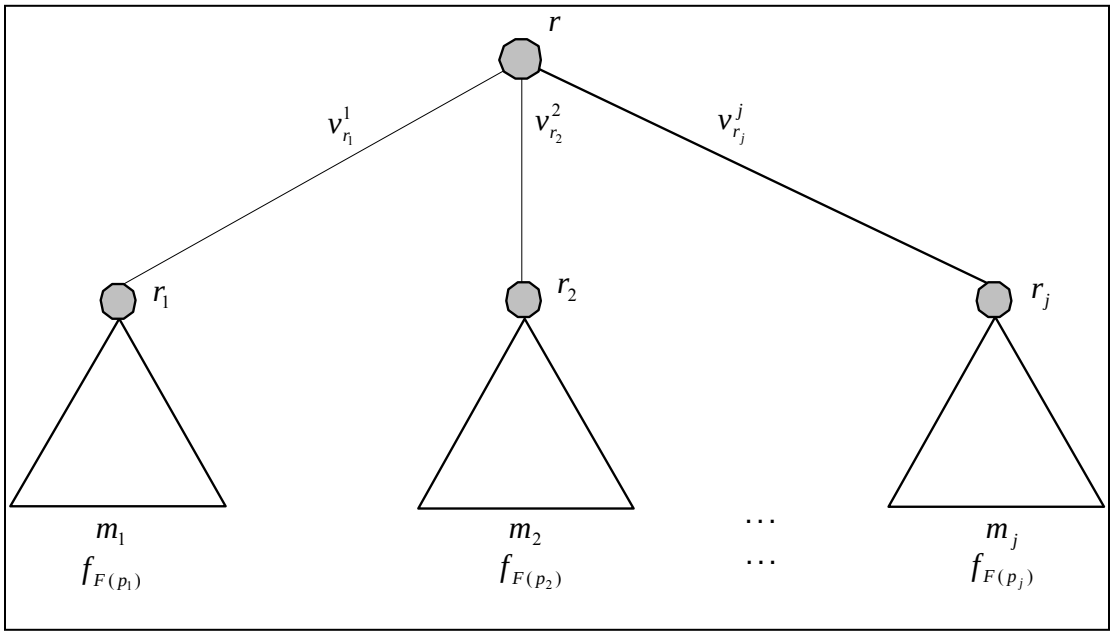

Figure 4.5: Fan $F(j)$ with feasible duplicating delay vectors

Lemma 4: For the hook-up fan H in Figure 4.5, the multicast tree assignment f_H given by $f_r(.)$ and $f_{F(P_i)}$, $1 \le i \le j$ is a feasible multicast tree assignment.

Proof: $f_{F(P_i)}$, $1 \le i \le j$ are feasible multicast fan assignments for $F(p_i)$.

$< v_{r_1}^1, v_{r_2}^2, ..., v_{r_j}^j >$ is a feasible duplicating delay vector

In the reminder of the section we will show that f_H is also an optimal multicasting tree assignment for H. We need a few results before that. Let H be a hook-up fan as shown in Figure 4.6 with an optimal multicasting tree assignments as indicated.

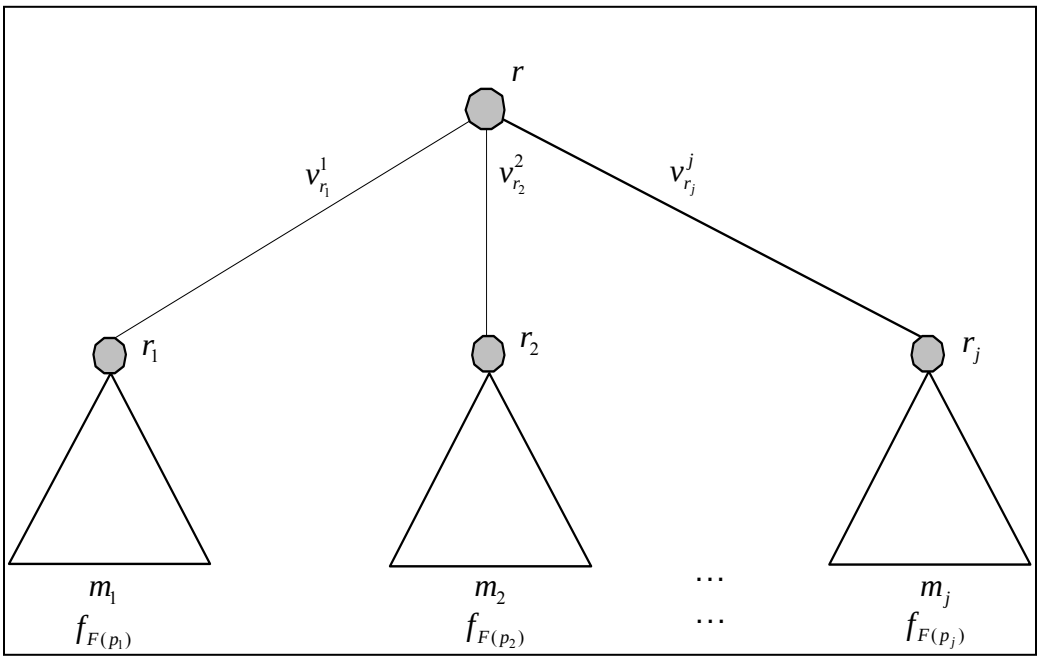

Figure 4.6: Hook-up fan *H* with optimal multicasting tree assignments

$$u_i = PD_{max}\,(\,f_{F(P_i)}\,),\ 1 \le i \le j.$$

Let $t_S + u_S = PD_{OPT}\,(H)$, where $s \in \{1, 2, \ldots\ldots, j\}$, without loss of generality. In other words, there could be more than one path from *r* with the same value for optimal delay. We break ties arbitrarily and pick one such indexed by. ■

Lemma 5: Given *H* as above, u_S is optimal for $F(P_S)$, i.e., $u_S = PD_{OPT}\,(F(P_S))$.

Proof: Suppose u_S is not optimal for $F(P_S)$. Then there exists an optimal assignment for $F(P_S)$ such that optimal multicasting duplicating delay for $F(S) = v_S = PD_{OPT}\,(F(P_S))$. Clearly, then $v_S < u_S$. It is clear, using this new assignment for $F(P_S)$, we could construct another feasible assignment for *H*. Let us call this new feasible assignment for *H*, $f_H^{OPT}\,[new]$. In $f_H^{OPT}\,[new]$, we have new values for the path delays originating

at r and ending at leaves of F_S. In particular the maximum path delay of $u_S + t_S$ becomes $v_S + t_S$. We know that $v_S + t_S < u_S + t_S$. Two possibilities exist for the optimal assignment of H.

(i) $v_S + t_S > u_i + t_i$, $i \neq s$, $1 \leq i \leq j$ or

(ii) $\exists\, q \in \{1, 2, \ldots \ldots, j\}$, $q \neq s$ such that

$$u_q + t_q > u_i + t_i \text{ , } 1 \leq i \leq j \text{ and } i \neq s \text{ and } u_q + t_q > v_s + t_s$$

In case (i) above, we have a new min max value $(v_S + t_S) < (u_S + t_S)$. And this is a contradiction. In case (ii) above, there is a new min max delay on a different path. In this case, $u_q + t_q > v_s + t_s$ and $u_q + t_q < u_s + t_s$. Hence $u_q + t_q$ is a maximum that is less than the optimal value $u_s + t_s$. Again this is a contradiction. ■

Lemma 5 is crucial because it suggests that we could have sub-optimal solutions for all but one fan and still get an optimal solution or assignment for a hook-up fan. The next lemma extends this idea and shows that any optimal assignment for a hook-up fan can be made to consist of optimal assignments for all fans in a hook-up fan.

Lemma 6: For a hook-up fan H, and an optimal multicast tree assignment f_H^{OPT}, there exists another optimal multicast tree assignment f_H^{OPT} [*new*] such that all the fans of H, $F(p_1)$, $F(p_2)$,…. …. , $F(p_j)$ have optimal assignments.

Proof: From Lemma 5, we know that there exists one fan $F(p_S)$ with an optimal assignment, where $s \in \{1, 2, \ldots \ldots, j\}$.Without loss of generality, let $F(p_q)$ be a fan which does not have an optimal assignment where $q \neq s$ and $q \in \{1, 2, \ldots \ldots, j\}$. Let PD_{OPT} (H) be the optimal multicasting duplicating delay for H. We know that PD_{OPT} (H) is of the form $u_s + t_s$ where u_s is the optimal value for $F(P_s)$. Hence $u_s + t_s > u_i + t_i$, $i \neq s$, $1 \leq i \leq j$. In particular $u_s + t_s > u_q + t_q$ where u_q is sub-optimal for $F(p_q)$. Let v_q be optimal for $F(p_q)$. Then $v_q < u_q$ and hence by substituting an optimal assignment for $F(p_q)$, we get a new assignment for H. The only change in the path delay is the value of the q'th path where $u_q + t_q$ changes to $v_q + p_q$. Since $u_s + t_s > u_q + t_q$, we have $u_s + t_s > v_q + p_q$. This means that the new optimal assignment preserves the optimal delay value $u_s + t_s$. Hence all suboptimal assignments for the fans can be replaced by optimal assignments without a change to the optimal value $u_s + t_s$. ■

Lemma 5 and 6 lead to the following important theorem.

Theorem 7: Given a hook-up fan H as in Lemma 4 with multicasting tree assignment f_H, f_H is an optimal multicasting tree assignment.

Proof: We know from Lemma 4, f_H is a feasible assignment for H made up of $f_r(.)$ and $f_{F(P_i)}$, $1 \leq i \leq j$. We also know that f_H is an optimal solution to the system of duplicating delay vectors (of the fan obtained from H).

$< t_1^i + m_i, t_2^i + m_i, \ldots\ldots, t_j^i + m_i >$, where $m_i = PD_{OPT}(F(p_i))$,

and $< t_1^i, t_2^i, \ldots\ldots, t_j^i > = D(r_i)$ of H, $1 \leq i \leq j$

From Lemma 6, we know that there exists an optimal solution for hook-up fan *H*, whose fans also have optimal multicasting tree assignments. This is shown in Figure 4.7.

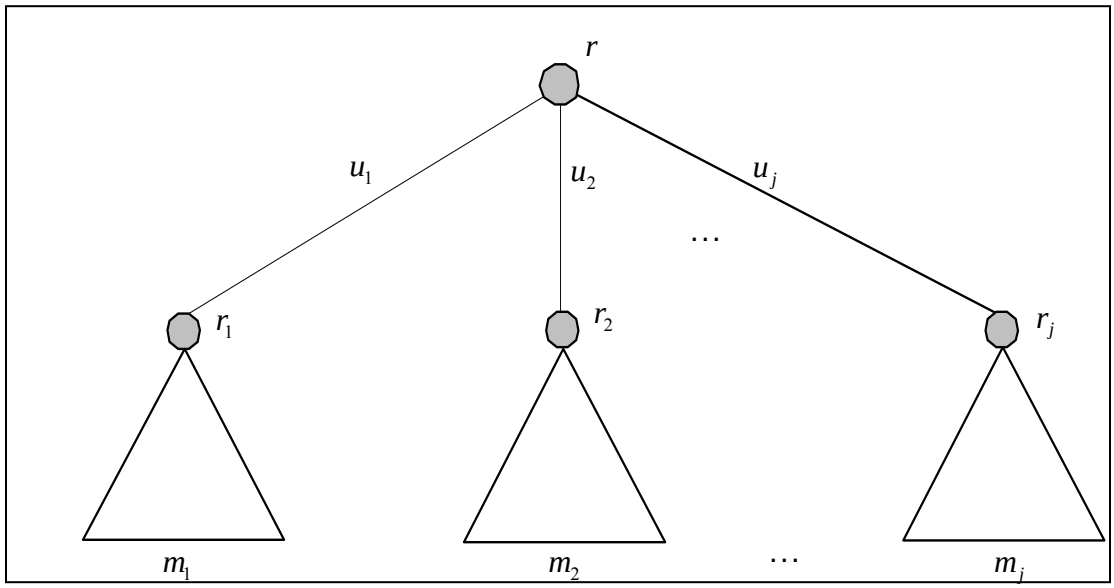

Figure 4.7: Hook-up fan *H* with optimal multicasting

where m_i is the maximum delay for fan $F(p_i)$ and m_i is optimal for $F(p_i)$, $1 \le i \le j$. The optimal duplicating delay for *H* is $max\ \{u_i + t_i : 1 \le i \le j\}$. Secondly, delays $\{u_i + t_i : 1 \le i \le j\}$ are an optimal solution to the same set of duplicating delay vectors $< t_1^i + m_i, t_2^i + m_i, ..., t_j^i + m_i >$, $1 \le i \le j$. Hence the theorem. ■

Theorem 7 tells us that we can obtain an optimal solution to a hook-up fan by a bottom-up approach. Any multicasting tree can be obtained by a series of hook-up operations starting from the base fans.

4.2.6 Example

Bottom-up approach to computing the optimal multicasting tree assignment using hook-up fan decomposition is shown in Figure 4.8.

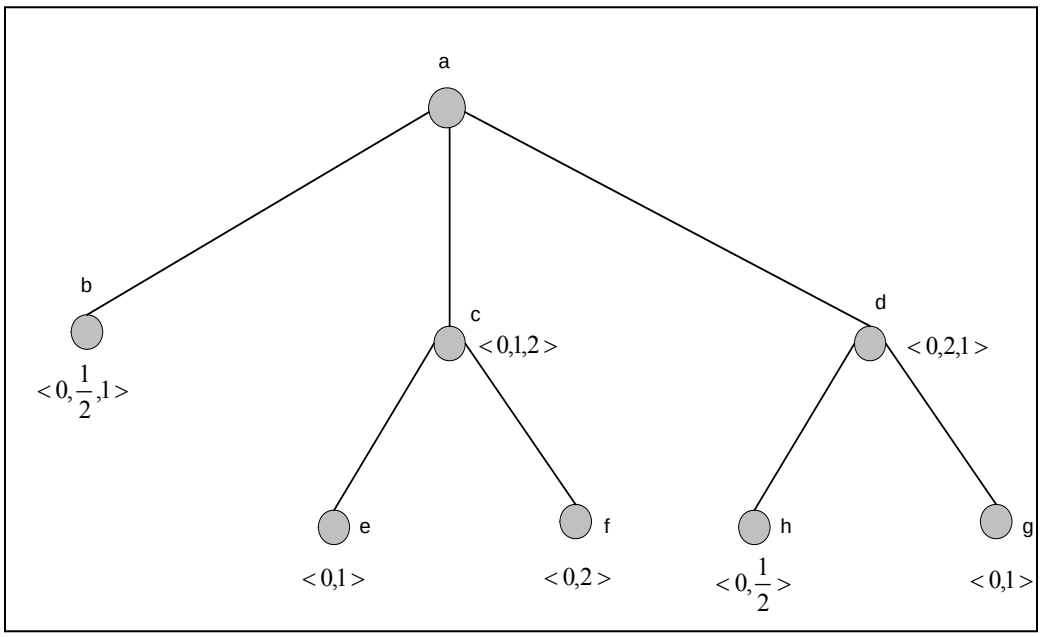

Figure 4.8: Multicast tree with duplicating delay vectors at nodes and assuming that the link delays are the same on all links.

The steps for computing the optimal solutions for fans from Figure 4.8 are shown in Figure 4.9 and Figure 4.10.

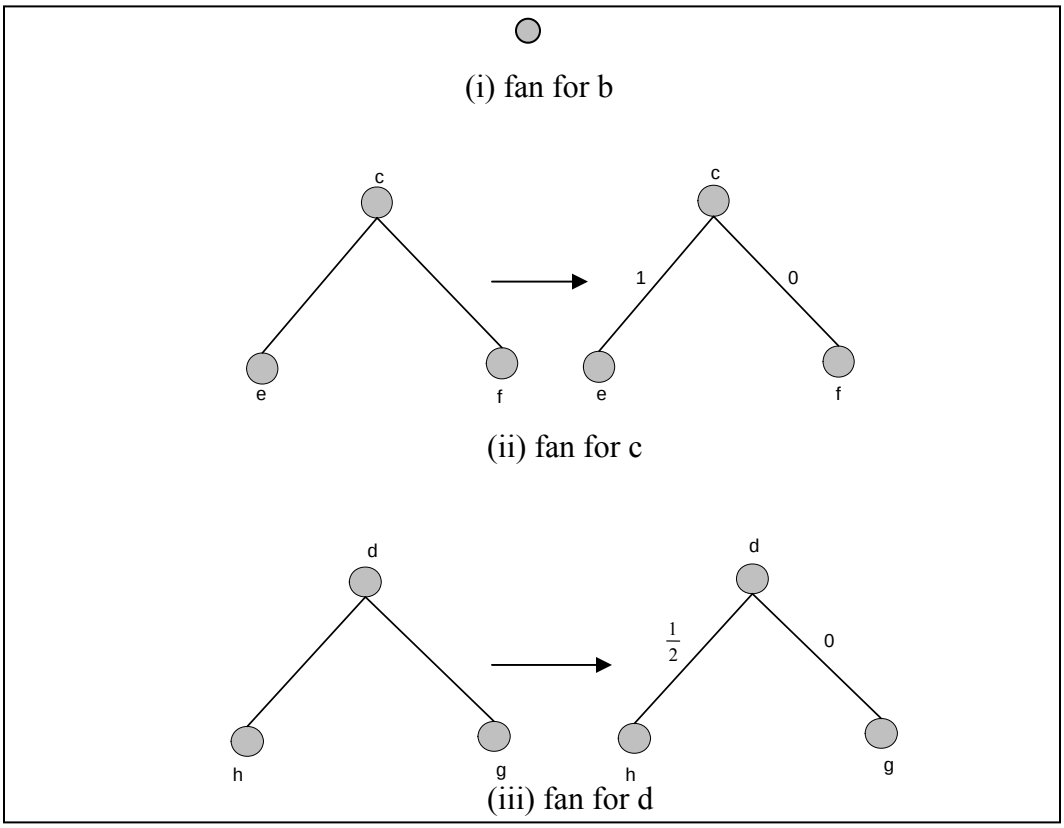

Figure 4.9: Optimal Solutions for fans b, c and d

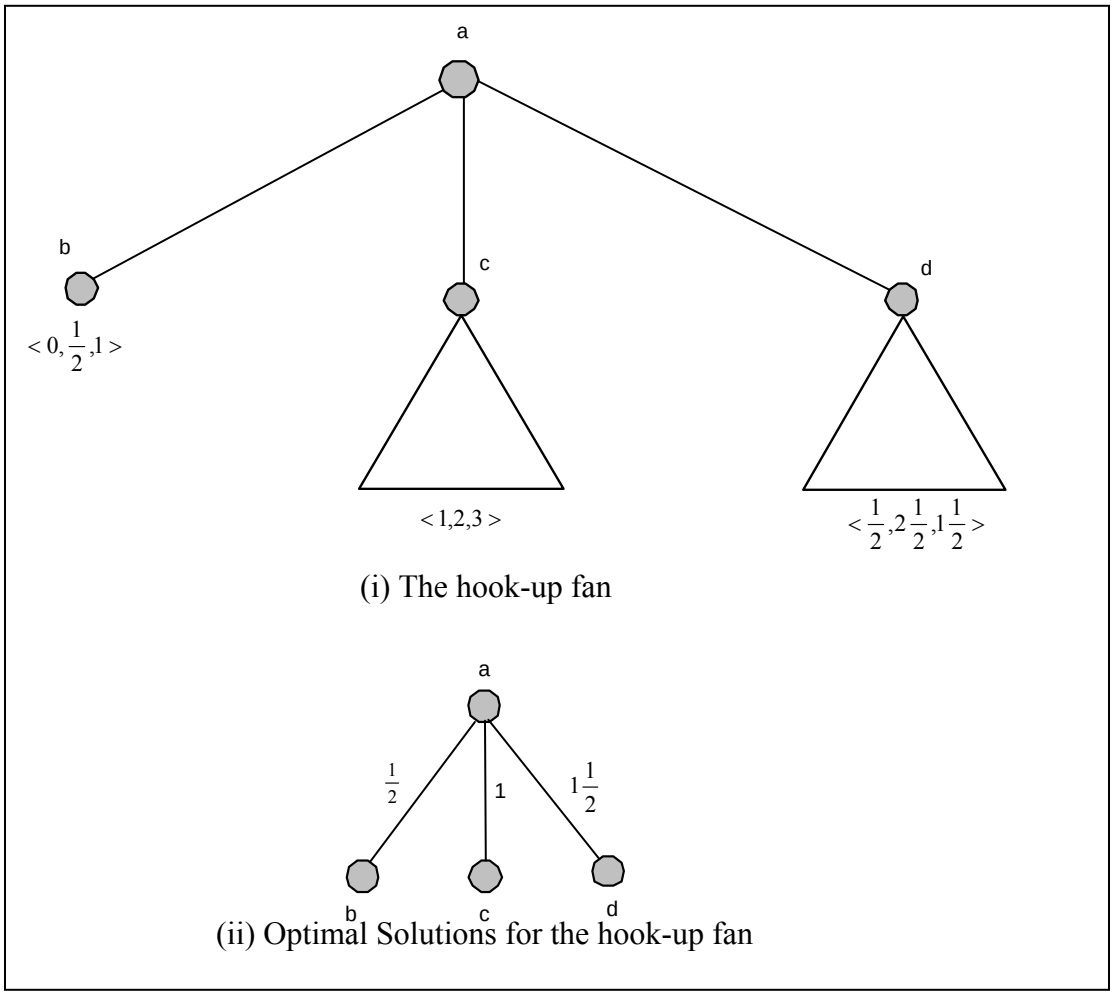

Figure 4.10: The Hook-up fans and its Optimal Solution

Re-working the optimal solutions we get the optimal multicast tree shown in Figure 4.11 with PD_{OPT} (T) = 1 ½ unit and the ordering at node 'a' is 'c', 'b' and 'd'. The ordering at node 'c' is 'f' and then 'e'. The ordering at node 'd' is 'h' and then 'g'.

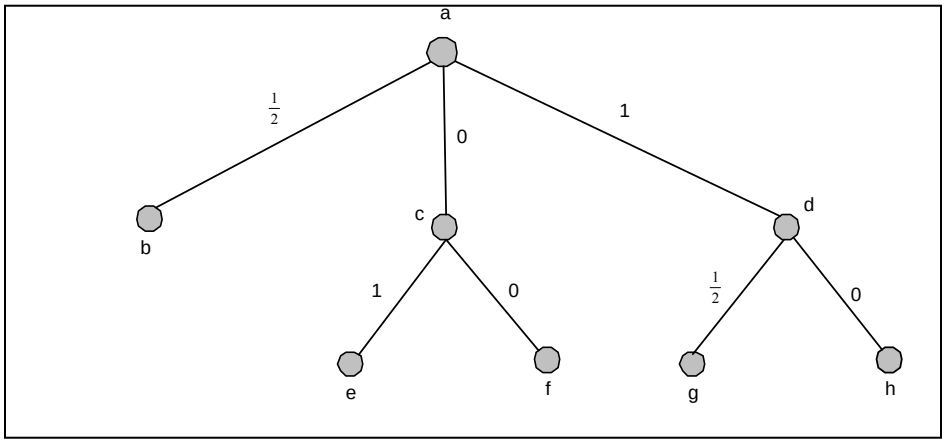

Figure 4.11: The optimal multicasting tree

4.3 Algorithm and its Time Complexity

4.3.1 Algorithm for Optimal Multicasting Delay

Repeat the following steps until the root of the tree is reached.

1. Find optimal solutions to base fans $F(p_i)$. Let $PD_{OPT}(F(p_i))$ be the delays.
2. Hook them up and add $PD_{OPT}(F(p_i))$ to duplicating delay vectors.
3. Find optimal solutions to hook-up fans with such modified duplicating delay vectors.

After the root is reached, re-work the solutions obtained top-down to get complete tree assignment.

4.3.2 Time Complexity of the Algorithm

In this section we address the time-complexity of our algorithm for optimal multicasting delay. For each fan $F(p_i)$, using Lemma 3, we can compute an optimal solution in $O(p_i^{5/2})$ time where p_i is the number of leaves in fan $F(p_i)$. During the bottom-up approach, let us say, we have a sequence $l_1, l_2, \ldots \ldots, l_j$ leaves when we get to the root where $l_1 + l_2 + \ldots \ldots + l_j = O(n)$. Hence the running time is bounded by

$$\sum_{i=1}^{j} O(l_i^{5/2}) \le (\sum_{i=1}^{j} l_i)^{5/2}$$

$$= O(n^{5/2})$$

Theorem 8: The optimal multicasting tree assignment problem can be solved in $O(n^{5/2})$ time.■

4.4 Lower Bound Result

From Lemma 3, we know that given a multicast fan T, a special case of a tree, an optimal multicast tree assignment for T and the corresponding optimal multicasting switching delay can be found in $O(n^{5/2})$ time, where n is the number of nodes in T. Conversely, we can also show in a straightforward fashion that solving the multicasting tree problem is at least as hard as the min-max matching problem. Hence, it is unlikely that the above time-complexity can be improved easily. To see this, let there be a

weighted, complete bipartite graph $G = (S, D, F)$ where $S = \{ v_1, v_2, ..., v_k\}$, $D = \{1, 2, ..., k\}$, and edge set $F = \{(v_i, j): 1 \le j \le k, 1 \le i \le k\}$. The weight of an edge $e = (v_i, j) \in F$ is given by $w((v_i, j)) = t_j^i$, $1 \le i, j \le k$.

We transform this graph into a fan $T = (V, E)$ which is a multicast tree with $k + 1$ nodes, where k of the $(k + 1)$ nodes are leaves attached directly to the root node. Let the leaf nodes be $v_1, v_2, ..., v_k$ attached to the root node that we call v. Let $D_i = w((v_i, j)) = t_j^i$ where $1 \le j \le k$ for each i, $1 \le i \le k$. Indeed, D_i can be taken to be the duplicating delay vector for node v_i.

Furthermore, it is easy to see that computing the min-max matching on G can be achieved by computing the optimal multicasting tree assignment T. Noting that solving the optimal multicasting tree assignment for an arbitrary tree is as hard as for a special case of fan, we have proved that the optimal multicasting tree assignment problem has a lower bound of $O(n^{5/2})$ time.

4.5 Simulation Experiments

To validate our argument that ordering is important in overlay multicasting, we have written a program for multicasting in the application layer using BSD sockets [22]. Our experimental setup consists of four nodes with one source and three destinations. Before data is multicast from the source to the destinations, the time of all the nodes

are synchronized with a centralized time-server. In our experiment, we have chosen the source node, time-server and one destination node at University of Oklahoma (OU), Norman Campus, one destination node at Loyola University Chicago (LUC) and one destination node at University of British Columbia (UBC) Canada as shown in Figure 4.12. The source multicasts data to the destinations and each destination computes the delay when it receives the data. We run the experiment with changing the order of multicasting at source. Each packet was send over 30 times and the average time taken is shown in the Table 4.1. The results are plotted in Figure 4.13. Our experimental results show that ordering changes the delay for packets at each destination.

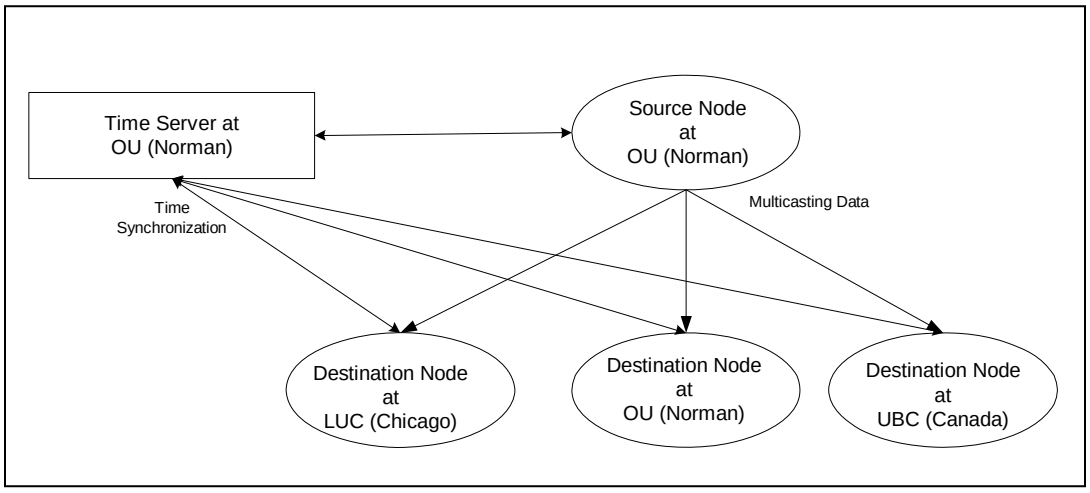

Figure 4.12: Experimental Setup for Multicasting Application

	Delay at each node in seconds of multicast packets of size 1kb		
Ordering	OU, Norman (A)	LUC,Chicago (B)	UBC, Canada (C)
A-B-C	0.145	1.522	1.769
A-C-B	0.1	1.585	1.606
B-C-A	0.428	1.383	1.448
B-A-C	0.09	1.367	1.502
C-A-B	0.202	1.548	1.504
C-B-A	0.337	1.396	1.485

Table 4.1: Table demonstrating that order matters in multicasting.

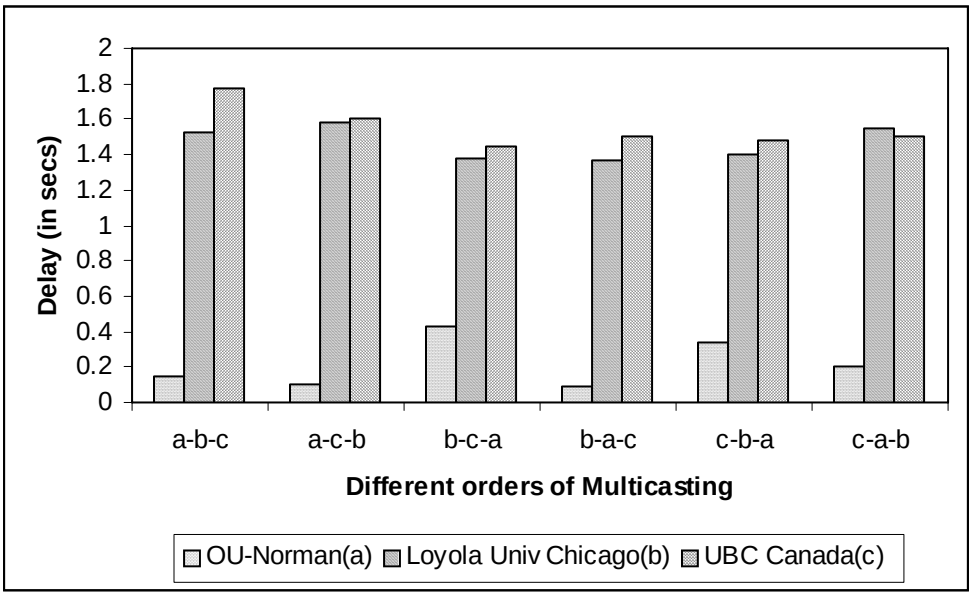

Figure 4.13: Plotting of Different Orders of Multicasting VS Delay

4.6 Summary

In this research, we have considered a more generalized form of switching delay vectors (called duplicating delay vectors) where all the elements of a vector may not be equal. Given a multicast tree with link delays and generalized delay vectors at each non-leaf node, we provide an algorithm which schedules the message delivery at each non-leaf node in order to minimize the delay of the multicast tree. Our algorithm, which has a complexity of $O(n^{5/2})$, uses the concept of min-max matching problem on bipartite graphs. We also show an important lower bound result that optimal multicast switching delay problem is as hard as min-max matching problem on bipartite graphs.

Chapter 5

Embedding Multicasting trees on the Overlay Network

5.1 Introduction

The problem of embedding multicasting trees on the overlay network can be reduced to delay constrained sub-graph homeomorphism problem. The sub-graph homeomorphism problem can be defined as follows: given a guest graph *G* and a host tree *H*, find whether *H* has a sub-graph *H'* that can be transformed to *G* by repeatedly removing any node of degree 2 and adding the edge joining the neighbors. In multicasting tree embedding problem, the guest graph is a tree $T = (V_T, E_T)$ where each link $e \in E_T$ is assigned a delay function $d(e)$ which specifies the maximum allowable delay along that link. This implies that if $e \in T$ is mapped to a path P in the overlay

network, the delay along the path P should be less than or equal to $d(e)$. Also we want that each edge $e \in T$ should be mapped to a shortest path in the overlay network. Based on the constraint of our embedding and the definition of homeomorphic embedding, we write following two important lemmas.

Lemma 1: Let G be a tree. If G is homeomorphic to a graph H, then the subgraph H' of H which is homeomorphic to G is a tree.

Proof: Suppose G is a tree which is homeomorphic to a graph H. Let H' be the subgraph of H which is homeomorphic to G. We have to prove that H' is a tree. According to the definition of homeomorphism, G can be obtained from H' by repeatedly removing any node of degree 2 and adding the edge joining its two neighbors. Suppose H' is not a tree and H' contains a cycle. It can be easily shown that any cycle with n nodes (C_n) can be contracted to a cycle with 3 nodes (C_3) which implies that C_3 is homeomorphic to C_n. C_3 cannot be contracted any further as it will create multiple edges. So the subgraph homeomorphic to C_3 will contain C_3. We have assumed that H' contains a cycle which can be contracted to C_3. If G is homeomorphic to H', then G contains C_3. This is a contradiction since G is a tree. ■

Lemma 2: Let G be a tree. If G is homeomorphic to a graph H in such a way that each edge of G is mapped to a shortest path in H, then the subgraph H' of H which is homeomorphic to G is a shortest path tree in H.

Proof: Suppose G is homeomorphic to a graph H in such a way that each edge of G is mapped to a shortest path in H. Let H' be the subgraph of H which is homeomorphic to G. From Property 1, we know that H' is a tree. We have to show that H' is a shortest path tree of H. Suppose H' is not a shortest path tree of H. Then there exists at least two nodes $u, v \in H$ for which the path $P(u\sim v)$ in H' is not the shortest path between nodes u and v in H. For the sake of generality, we assume that u is an ancestor of v in H'.

Case (i): An edge $(x, y) \in G$ is mapped to path $P(u\sim v)$ in H'. Since G is homeomorphic to H', the path $P(u\sim v)$ in H' is the shortest path between node u and v in H. This is a contradiction.

Case (ii): Nodes u and v are included in a shortest path between nodes p and q (p ancestor of q) in H' and an edge $(x, y) \in G$ is mapped to path $P(p\sim q)$ in H'. Since any subpath of a shortest path is also a shortest path, path $P(u\sim v)$ in H' is a shortest path between nodes u and v in H. This is a contradiction.

Case (iii): Node v is included in the shortest path between nodes p and q (p ancestor of q) in H' but node u is not included in that path and an edge $(x, y) \in G$ is mapped to path $P(p\sim q)$ in H'. Then path $P(v\sim p)$ is the shortest path between nodes v and p in H. Suppose an edge $(x, z) \in G$ is mapped to path $P(p\sim r)$ in H' and node u is contained in $P(p\sim r)$. Then path $P(p\sim u)$ in H' will be the shortest path between node p and u in H. This implies that $P(u\sim v)$ in H' is the shortest path between node u and v in H. This is a contradiction. ■

Lemma 1 and Lemma 2 suggests that we search all the shortest path trees of H to find out whether delay constraint homeomorphic (shortest path) embedding of G_T exists in H. Now, given a multicast tree G_T and a shortest path tree H_T in the overlay network, our problem is to find whether H_T has a sub-tree homemorphic to G_T such that the delay constraint for each edge of G_T is satisfied. We need to do this process for all shortest path trees of H.

The following example illustrates our problem. In Figure 5.1, we have a shortest path tree H_T and a multicasting tree G_T. The edge (x, z) of G_T can be mapped to edge (A, C) of H_T and $d(\text{A, C}) \leq d(x, z)$. Similarly edges (y, s) and (y, t) of G_T can be mapped to edges (D, E) and (D, F) of H_T respectively and $d(D, E) \leq d(y, s)$ and $d(D, F) \leq d(y, t)$. Now edge (x, y) of G_T can be mapped to path (A-B-D) of H_T and $d(\text{A-B-D}) \leq d(x, y)$. So H_T has a sub-tree which is homeomorphic to G_T and the homeomorphism satisfies the delay constraint of G_T.

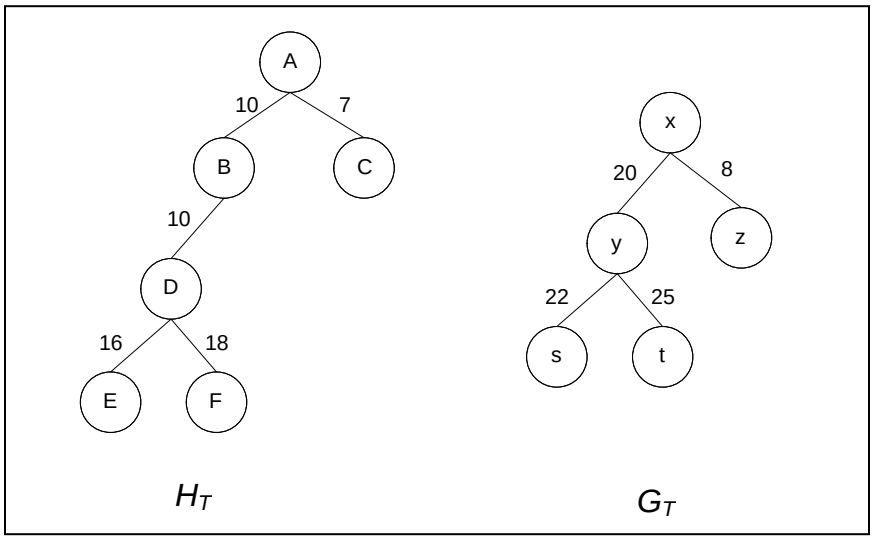

Figure 5.1: A shortest path tree H_T and a multicasting tree G_T

5.2 Formal Definition of the Problem

Suppose we have a host tree $H_T = (V_H, E_H)$ where V_H denotes the set of NSNs (or end-users) and E_H denotes the set of links in H_T. For each link $e_h \in E_H$, we have a delay function D_H: $E_H \rightarrow \Re^+$ which assigns end-to-end delay along each link in E_H. We have a guest tree (multicasting tree) $G_T = (V_G, E_G)$ where V_G is the set of nodes specified by the collaborative application desginer and E_H is the set of virtual links connecting the nodes in G_T. For each link $e_g \in E_G$, we have a delay function D_G: $E_G \rightarrow \Re^+$ which denotes the maximum allowed end-to-end delay along each virtual link in E_G.

The tree G_T can be embedded in H_T when there exists an injective mapping f which maps the nodes of G_T to the nodes of H_T and for every pair of nodes u_G, v_G of G_T, if there is a link (u_G, v_G) in G_T, then there exists a path from $f(u_G)$ to $f(v_G)$ in H_T with all intermediate nodes of out-degree 1 and with no intermediate node in G_T and the delay of the path $f(u_G) \sim f(v_G)$ is less than or equal to the delay of the link (u_G, v_G) in G_T which implies that $d(f(u_G) \sim f(v_G)) \leq d(u_G, v_G)$.

Given a guest tree G_T and a host tree H_T with delay functions D_G and D_H defined on G_T and H_T, respectively, find whether H_T has a sub-tree H_t which is homemorphic to G_T by repeatedly removing any node of degree 2 and adding the link joining its two neighbors such that the delay constraint for each link in G_T is satisfied in H_t.

5.3 Our approach

The sub-graph homeomorphism problem which determines for given two graphs G and H whether H has a sub-graph H' homeomorphic to G is an NP Complete problem [61]. But the sub-tree homeomorphism problem is polynomial-time solvable and different algorithms are proposed for this problem in the literature which are defined for un-weighted trees. The most efficient algorithm for sub-tree homeomorphism problem on un-weighted trees has been proposed by [26] which has a complexity of $O(n^{2.5})$ where n is the number of nodes in the host tree. In our solution, we modify the algorithm of [26] such that it determines whether the guest tree G_T is delay constrained homeomorphic to the host tree H_T. For the sake of simplicity we will denote the host tree with H and the guest tree with G.

5.3.1 Terminologies

We describe the algorithm for delay constrained sub-tree homeomorphism problem for a rooted tree. A host tree $H = (V_H, E_H)$ with root node r is called a rooted tree and is denoted as $H_r = (V_H, E_H, r)$. The rooted tree specifies the direction for each link which points away from the root. The sub-tree generated by node v in H_r is denoted as $H_r(v)$. If the guest tree is rooted at r', then it can represented by $G_{r'} = (V_G, E_G, r')$. For each node $v \in V_H$ in H_r , we define the following two terms:

$DS^r(v) = \{x \in V_G \mid$ there is sub-tree of $H_r(v)$ which is delay constrained homeomorphic to $G_{r'}(x)\}$.

$P^r(v) = \{x \in V_G \mid x$ can be potentially embedded on $v\}$.

Now if we can show that $r' \in DS^r(v)$ which implies that the sub-tree rooted at v in H_r has a delay constrained sub-tree homeomorphic to $G_{r'}$, then we have solved our problem. $DS^r(v)$ and $P^r(v)$ for each node $v \in V_H$ in H_r is computed in bottom-up fashion. Initially $DS^r(v)$ and $P^r(v)$ is computed for all the leaf nodes in H_r and the leaf nodes are marked. Note that all the leaf nodes in $G_{r'}$ will be included in $DS^r(v)$ and $P^r(v)$ if v is a leaf node in H_r. Next, $DS^r(v)$ and $P^r(v)$ is computed for a non-leaf node v in H_r, if $DS^r(w)$ is computed for the all the children w of v in H_r. It can be easily proved that if $u \in DS^r(w)$ and v is the parent of w, then $u \in DS^r(v)$. It can also be easily proved that if $u \in G_{r'}$ is a leaf node, then $u \in P^r(v)\ \forall\ v \in H_r$.

Suppose node v in H_r has children $x_1, x_2, \ldots, x_s$, and node u in $G_{r'}$ has children $y_1, y_2, \ldots, y_t$ and x_i and y_j are the leaf nodes of H_r and $G_{r'}$, respectively for $1 \le i \le s$ and $1 \le j \le t$. Now we need to decide whether $u \in DS^r(v)$. This problem can be solved by a bipartite matching problem. We construct a bipartite graph B with partitions X and Y where X is the set of children of v in H_r and Y is the set of children of u in $G_{r'}$, and make an edge (x_i, y_j) in B iff $y_j \in P^r(x_i)$ and $d(v, x_i) \le d(u, y_j)$ where $d(v, x_i)$ and $d(u, y_j)$ are the delay of the edges (v, x_i) and (u, y_j) in H_r and $G_{r'}$, respectively. We compute the

matching M of the bipartite graph B. If $|M| = |Y|$, then we include u in $DS^r(v)$ and in $P^r(v)$ and mark v. This is illustrated with an example in Figure 5.2.

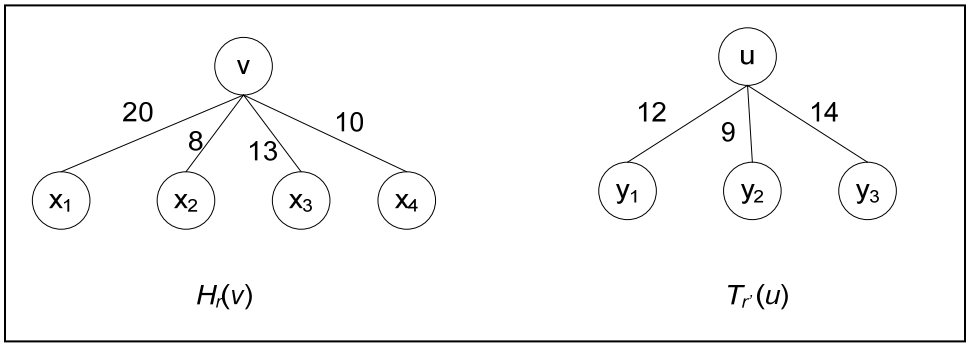

Figure 5.2: An example of $H_r(v)$ and $T_{r'}(u)$

Initially we compute the $DS^r(\ )$ and $P^r(\ )$ for all the leaf nodes in $H_r(v)$. $DS^r(x_1) = \{y_1, y_2, y_3\}$, $DS^r(x_2) = \{y_1, y_2, y_3\}$, $DS^r(x_3) = \{y_1, y_2, y_3\}$, $DS^r(x_4) = \{y_1, y_2, y_3\}$, $P^r(x_1) = \{y_1, y_2, y_3\}$, $P^r(x_2) = \{y_1, y_2, y_3\}$, $P^r(x_3) = \{y_1, y_2, y_3\}$, and $P^r(x_4) = \{y_1, y_2, y_3\}$ and mark the leaf nodes x_1, x_2, x_3 and x_4 as mapped. Next we construct the bipartite graph B with $X = \{x_1, x_2, x_3, x_4\}$ and $Y = \{y_1, y_2, y_3\}$. Edge (x_2, y_1) will be included in B as $y_1 \in P^r(x_2)$ and $d(v, x_2) \leq d(u, y_1)$. Similarly edges (x_2, y_2), (x_2, y_3), (x_3, y_3), (x_4, y_1) and (x_4, y_3) will be included in B. The matching M of B will be $\{(x_3 - y_3), (x_2 - y_2), (x_4 - y_1)\}$ and $|M| = |Y|$. So u is included in $DS^r(v)$ and $P^r(v)$ and v is marked as mapped.

Next, we generalize $DS^r(v)$ and $P^r(v)$ for any non-leaf node v in H_r based on the following condition. Let v be a node in H_r with children $x_1, x_2, \ldots, x_s$, and u be a node in $G_{r'}$ with children $y_1, y_2, \ldots, y_t$ and $DS^r(x_i)$ for each child x_i of v for $1 \leq i \leq s$ is computed. Now we construct the bipartite graph B with X and Y where $X = \{x_1, x_2, \ldots, x_s\}$ and $Y = \{y_1, y_2, \ldots, y_t\}$. In B, an edge is created between x_i and y_j,

i) If $y_j \in P^r(x_i)$ and $d(v, x_i) \leq d(u, y_j)$ OR

ii) if $y_j \notin P^r(x_i)$ but there exists a node x_{ik} along the path ($v - x_i - child(x_i) - child(child(x_i))) - \ldots - x_{ik}$) where $y_j \in P^r(x_{ik})$ and $d(v - x_i - child(x_i) - child(child(x_i))) - \ldots - x_{ik}) \leq d(u, y_j)$.

5.4 Algorithm and its Time Complexity

5.4.1 Algorithm

The algorithm for finding whether a given host tree has a subtree which is delay constrained homeomorphic to a given guest tree is given below:

Algorithm Find_Delay_Constrained_Homeomorphism

Input: Rooted trees $H_r = (V_H, E_H, r)$ and $G_{r'} = (V_G, E_G, r')$ with delay functions D_H and D_G for H_r and $G_{r'}$, respectively.

Output: Yes if H_r has a sub-tree which is delay constrained homeomorphic to $G_{r'}$.

Begin

Initially all the nodes are not marked

For each leaf node v of H_r

$DS^r(v) = \{x \mid x$ is a leaf node of $G_{r'}\}$

$P^r(v) = \{x \mid x$ is a leaf node of $G_{r'}\}$

End for

For each node v of H_r

$P^r(v) = \{x \mid x$ is a leaf node of $G_{r'}\}$

End for

For each node v of H_r

If all the children of v are marked

Compute_ $DS^r(v)$_ $P^r(v)$

Mark node v

End if

End for

If $r' \in DS^r(r)$

return Yes

Else

return No

End if

End

Compute_ $DS^r(v)$_ $P^r(v)$

Begin

$DS^r(v) = \bigcup_{x \in X} DS^r(x)$ where X is the set of children of v

For each node u of $G_{r'}$

Construct the Bipartite graph B with X and Y where X is the set of children of v and Y is the set of children of u.

For each $x_i \in X$ and each $y_j \in Y$

If $y_j \in P^r(x_i)$

If $d(v, x_i) \leq d(u, y_j)$

Create an edge (x_i, y_j)

End if

Else

If (there exists a node x_{ik} along the path $(v - x_i - child\ (x_i) - child(child(x_i))) - \ldots - x_{ik})$ where $y_j \in P^r(x_{ik})$ and $d(v - x_i - child\ (x_i) - child(child(x_i))) - \ldots - x_{ik}) \leq d(u, y_j)$)

Create an edge (x_i, y_j)

End if

End if

End for

Compute matching M of B

If $|M| = |Y|$

Include u in $DS^r(v)$ and in $P^r(v)$

Mark v as mapped

End if

End for

End

Theorem: The algorithm Find_Delay_Constrained_Homeomorphism is correct.

Proof: In the algorithm Find_Delay_Constrained_Homeomorphism, a node $u \in G_{r'}$ is included in $DS^r(v)$ and $P^r(v)$ of a node $v \in H_r$, only when all the children of u are feasibly mapped to the children of v. This is done with the bipartite matching. An edge between x_i (a child of v) and y_j (a child of u) is created in the bipartite graph, if x_i is feasibly mapped to y_j and the delay of (v, x_i) is less than or equal to the delay of (u, y_j). If x_i is not feasibly mapped to y_j, then the algorithm traverses the tree downwards to find out the node which is feasibly mapped to y_j. This exhaustive traversal takes care of the possible feasible embedding of an edge in $G_{r'}$ to a path in H_r. ■

5.4.2 Time Complexity

The algorithm Find_Delay_Constrained_Homeomorphism decides whether H_r has a sub-tree homeomorphic to $G_{r'}$ or not. We assume $|V_H| = n$ and $|V_G| = m$. Now we compute the complexity of $DS^r(v)$ and $P^r(v)$ computation of each $v \in H_r$. Suppose t_i and s_i be the number of children of node u_i in $G_{r'}$ and node v_i in H_r, respectively. We can write $\sum_{i=1}^{n} s_i = n-1$ and $\sum_{i=1}^{m} t_i = m-1$. From [57], we know that the matching problem on a bipartite graph with node partition of size t and size s can be solved in time $ct^{3/2}s$ for some constant c. Hence the time complexity of computing $DS^r(v)$ and $P^r(v)$ will be bounded by $\sum_{j=1}^{m} c s_i t_j^{1.5} \le c\, s_i m^{1.5}$. We compute $DS^r(v)$ and $P^r(v)$ for each node $v_i \in V_H$ and

$|V_H| = n$. Also when we construct the edges in the bipartite graph, if $y_j \notin P^r(x_i)$, we traverse down the tree H_r and find out the node for which y_j belongs to its $P^r(\ )$ value. The complexity of this operation is O(n). Hence, the total complexity of Find_Delay_Constrained_Homeormorphism is $\sum_{i=1}^{n} c\, s_i\, m^{1.5}\, n \le c\, m^{1.5}\, n^2$. In the worst case, m will be equal to n. The total complexity of the algorithm is O($n^{3.5}$).

5.5 Summary

In this research, we have considered the problem of embedding a designer specified multicasting tree on the overlay network. We use the concept homeomorphic embedding of trees and propose an algorithm which determines whether a host tree is delay constrained homeomorphic to a guest tree. The complexity of our algorithm is O($n^{3.5}$).

Chapter 6

Conclusion and Future Work

6.1 Contribution

In this research, we have developed protocols and designed networks for computer supported group collaborative applications on overlay networks.

- We have defined floor control problem and analyzed different approaches for floor control on overlay networks. We have provided an algorithm for constructing an efficient communication channel for implementing distributed floor control protocols on overlay networks. We have proposed to implement and evaluate MAC protocols for LANs on the communication channel to resolve floor control among the end-users. Towards this, we have considered

two well known MAC protocols: ALOHA and DQDB and showed how these protocols can be efficiently implemented on the communication channel constructed from the overlay network. We have described each protocol using algorithms and state diagrams. We have proved that implementation of our distributed floor control protocols preserves causal ordering of messages. Using an analytical model, we have derived the efficiency of each protocol. We have also performed simulation experiments for each protocol. Both the analytical and simulation results show that implementation of DQDB as a floor control protocol on overlay networks outperforms the implementation of ALOHA for the same purpose in terms of efficiency.

- We have considered the problem of determining a multicasting sub-network with end-to-end delay bound and delay variation bound. We have discussed three well-known heuristics from the literature and exposed their limitations. Then we have presented our heuristic Chains, which achieves the tightest delay variation for a given delay bound. We have implemented all the heuristics and observed Chains outperforms DPDVB and DVMA in terms of execution time. The Chains heuristic also achieves the tightest delay variation bound along with DPDVB. We have also presented results to show that finding *k*-shortest paths for all destinations is not a bottleneck in our solution. For dynamic reorganization of multicasting sub-network with the tightest delay variation and

bounded delay, we notice that solution with Chains is more efficient than that of DVMA in terms of time-complexity.

- For overlay multicasting tree models, we have considered a more generalized form of switching delay vectors (called duplicating delay vectors) where all the elements of a vector may not be equal. Given a multicast tree with link delays and generalized delay vectors at each non-leaf node, we provide an algorithm which schedules the message delivery at each non-leaf node in order to minimize the delay of the multicast tree. Our algorithm, which has a complexity of $O(n^{5/2})$, uses the concept of min-max matching problem on bipartite graphs. We also show an important lower bound result that optimal multicast switching delay problem is as hard as min-max matching problem on bipartite graphs.

- We have considered the problem of embedding a collaborative application designer specified multicasting tree on the overlay network. In the multicasting tree embedding problem, an edge in the multicasting tree can be mapped to a path in the overlay network. Our problem is similar to *sub-tree homeomorphism* problem. Given two trees G_T and H_T, the *sub-tree homeomorphism* problem is to find whether H_T has a sub-tree H_t that can be transformed into G_T by repeatedly removing any node of degree 2 and adding the edge joining its two neighbors. In multicasting tree embedding problem, we have to ensure that the embedding satisfies the delay constraint on each link in the multicasting tree. In

this research, we have proposed an algorithm for delay constrained multicasting tree embedding problem on the overlay network.

6.2 Future Scope of Work

- Collaborative applications such as video-conferencing, multi-party games, distributed database replication and interactive simulations are group multicasting or multiple source multicasting by nature which implies that every member of the multicasting group may transmit data to other members in the group. There are two traditional approaches for group multicasting: source-based tree and shared tree. In the source-based tree approach, separate multicasting trees rooted at each source node are used. In the shared tree approach, a single tree referred as Core Based Tree (CBT) is used for multicasting. These two approaches exhibit contrasting behavior in terms of performance, overhead, scalability and robustness. Source-based tree achieves better end-to-end delay performance with high protocol overhead. This approach is not scalable when the group size is large. But source-based tree performs better in terms of fault tolerance. Shared tree approach suffers the drawback of network congestion as data from each source node traverse the edges of the shared tree. This approach is not fault tolerant and increases end-to-end delay in data transmission. The advantages of

shared tree approach are it has less protocol overhead and it is scalable. As part of our future research, we want to study the multiple source multicasting problems on overlay networks. Multicasting with end-to-end delay bound and delay variation bound for group multicasting on overlay networks has not been addressed in the literature. We want to analyze both the approaches for collaborative applications on overlay networks.

- Network protocol designers use different simulation tools to experiment the efficiency, throughput of their protocols. The main strength of simulation is that the researchers can control the parameters of simulation environment. But this is not a realistic model of the real Internet. Moreover critiques often question the model being too artificial and simple. To overcome these deficiencies and to provide greater realism in the experiments, wide area distributed testbeds (such as Planetlab [62]) are getting popular among the network researchers. Using the widely deployed distributed nodes in the Planetlab testebed, researchers can test their protocols. A researcher needs to select a subset of nodes from the testbed that satisfy the parameters of a given testbed configuration. As part of our future research, we want to implement the proposed protocols on virtual testbeds on the Internet to analyze their actual performance.

Bibliography:

[1] Sherlia Y. Shi, Jonathan. S. Turner, "Multicast Routing and Bandwidth Dimensioning in Overlay Networks," IEEE Journal on Selected Areas in Communication, Vol. 20, Issue 8, October 2002, pp. 1444-1455.

[2] Sherlia Y. Shi, Jonathan. S. Turner, "Routing in Overlay Networks," IEEE INFOCOMM 2002, June 2002, pp. 1200-1208.

[3] Eli Brosh, Yuval Shavitt, "Approximation and Heuristic Algorithms for Minimum Delay Application-Layer Multicast Trees," IEEE INFOCOMM 2004, March 2004.

[4] Suman Banerjee, Christopher Kommareddy, Koushik Kar, Bobby Bhattacharjee, Samir Khuller, "Construction of an Efficient Overlay Multicast Infrastructure for Real-time Applications," IEEE INFOCOMM 2003, March 2003.

[5] Anton Riabov, Zhen Liu, Li Zhang, "Overlay Multicast Trees of Minimal Delay," 24th IEEE International Conference on Distributed Computing Systems ICDCS 2004, 2004.

[6] Yang-hua Chu, Sanjay G. Rao, Srinivasan Seshan, Hui Zhang, "Enabling Conferencing Applications on the Internet using an Overlay Multicast Architecture," ACM SIGCOMM 2001, August 2001, pp. 55-67.

[7] Napster website: http://www.napster.com

[8] Gnutella website: http://gnutella.wego.com

[9] H. Erikson, "MBONE: The Multicast Backbone," Communication of the ACM, August 1994, pp. 54-60.

[10] Yu-shun Wang, J. Touch, "Application Deployment in Virtual Networks using the X-Bone," DARPA Active Networks Conference and Exposition, May 2002, pp. 484-491.

[11] H. Peter. Dommel, J.J. Garcia-Luna-Aceves, "Efficacy of Floor Control Protocols in Distributed Multimedia Environment," Cluster Computing Journal, Special Issue on Multimedia Collaborative Environments, Vol. 2, No1,1999.

[12] R. Yavatkar, K.Lakshman, "Communication Support for Distributed Collaborative Applications," ACM / Springer Verlag Journal on Multimedia Systems, Vol. 2, No 2, 1994.

[13] K. Katrinis, G. Parissidis, B. Plattner, "Activity Sensing Floor Control in Multimedia Collaborative Applications," 10th International Conference on Distributed Multimedia Systems, San Francisco Bay, September, 2004.

[14] R. Qiu, F. Kuhns, J. R. Cox, "A Conference Control Protocol for Highly Interactive Video-conferencing," IEEE Global Telecommunications Conference, November 2002, pp. 2021-2025.

[15] J. Lennox, H. Schulzrinne, "A Protocol for Reliable Decentralized Conferencing," 13th International Workshop in Network and Operating Systems, Monterey, California, June 2003.

[16] Marek Wojciechowski, "Distributing and Replicating Data in Hospital Information Systems," Proceedings of the 5th International Conference on Computers in Medicine, Lodz, Poland, 1999, pp. 34-37.

[17] George N. Rouskas, Ilia Baldine, "Multicasting Routing with End-to-End Delay and Delay Variations Constraints," IEEE Journal on Selection Areas in Communications, Vol. 15, No 3, 1997, pp. 346-356.

[18] Kapoor Sanjiv, Raghavan Srivatsan, "Improved Multicast Routing with Delay and Delay Variation Constraint," Global Telecommunications Conference, 2000 (GLOBECOM '00), IEEE, Vol. 1, pp 476-480.

[19] Sheu Pi-Rong, Chen Shan-Tai, "A Fast and Efficient Heuristic Algorithm for the Delay and Delay Variation Bound Multicast Tree Problem," 15th International Conference on Information Networking, ICOIN'01, February 2001, Oita, Japan, pp 611-618.

[20] Jimenez M. Victor, Marzal Andres, "Computing the K Shortest Paths: A New Algorithm and an Experimental Comparison," Proceedings of 3rd Workshop, Algorithm Engineering, Lecture Notes in Computer Science 1668, Springer-Verlag 1999, pp 15-29.

[21] Jimenez Victor, Marzal Andres, "Algorithms for Computing the K Shortest Paths," http://terra.act.uji.es/REA/.

[22] W. Richard Stevens, "UNIX Networking Programming," Prentice Hall, 1998. pp. 48-49.

[23] A. S. Tanenbaum, Computer Networks, 3rd edition, Prentice Hall, Upper Saddle River, NJ, 1996.

[24] Garey M. R., Graham R. L., Johnson D. S., "The Complexity of Computing Steiner Minimal Trees," SIAM Journal of Applied Mathematics, Vol. 32, June 1977, pp 835-859.

[25] Francesc Rossello, Gabriel Valiente, "An Algebraic View of the Relation between Largest Common Subtrees and Smallest Common Supertrees," Theoretical Computer Science, 2004.

[26] Moon Jung Chung, "$O(n^{2.5})$ Time Algorithms for the Subgraph Homeomorphism Problem on Trees," Journal of Algorithms, Vol. 8, No. 1, 1987,pp.106-12.

[27] K. Birman, A. Schiper, P. Stephenson, "Lightweight Causal and Atomic Group Multicast," ACM Transactions on Computer Systems, Vol. 9, No 3, August 1991, pp. 272-314.

[28] M. Kalantar, K. Birman, "Causally Ordered Multicast: the Conservative Approach," Proceedings of the 19th International Conference on Distributed Computing Systems, June 1999, pp 36-44.

[29] S. E. Goodman, S. T. Hedetniemi, "On Hamiltonian Walks in Graphs", SIAM Journal on Computing, Vol. 3, No. 3, September 1974, pp. 214-221.

[30] D. W. Wall, "Mechanisms for Broadcast and Selective Broadcast," PhD Dissertation Stanford University, June 1980.

[31] S. M. Hedetniemi, E. J. Cockayne, S. T. Hedetniemi, "Linear Algorithms for Finding the Jordan Center and Path Center of a Tree," Transportation Science, Vol. 15, Issue 2, May 1981, pp. 98-114.

[32] A. Rosenthal, J. A.Pino, "A Generalized Algorithm for Centrality Problems on Trees," Journal of Association for Computing Machinery, Vol. 36, No. 2, April 1989, pp. 349-361.

[33] D. G. Thaler, C. V. Ravishankar, "Distributed Center-Location Algorithms," in IEEE Journal on Selected Areas in Communication, Vol. 15, Issue 3, April 1997, pp. 291-303.

[34] S. C. Bruell, S. Ghosh, M. H. Karaata, S. V. Pemmaraju, "Self-stabilizing Algorithms for Finding Centers and Medians of Trees," SIAM Journal of Computing, Vol. 29, Issue 2, 1999, pp. 600-614.

[35] Shankar M. Banik, Sridhar Radhakrishnan, Chandra N. Sekharan, "Multicast Routing with Delay and Delay Variation Constraints for Multimedia Applications," 7th IEEE International Conference on High Speed Networks and Multimedia Communications (HSNMC 2004), June 2004, pp. 399-411.

[36] Kabada Bharath-Kumar, Jeffrey M. Jaffe, "Routing to Multiple Destinations in Computer Networks," IEEE Transactions on Communications, Vol. Com-31, No. 3, March 1983, pp. 343-351.

[37] Liang Guo, Ibrahim Matta Matta, "QDMR: An Efficient QoS Dependent Multicast Routing Algorithm," Proceedings of the 5th IEEE Real-Time Technology and Applications Symposium, 1999, pp. 213-222.

[38] A. Hac, K. Zhou, "Multicasting Algorithm in Multimedia Communication Network," IEEE GLOBECOM'98, 1998, pp. 747-752.

[39] V. Kompella, J.C. Pasquale, G.C. Polyszos, "Multicast Routing for Multimedia Communications," IEEE/ACM Transactions on Networking, Vol. 1, No. 3, 1993, pp. 286-292.

[40] Mehrdad Parsa, Qing Zhu, J.J. Garcia-Luna-Aceves, "An Iterative Algorithm for Delay-Constrained Minimum-Cost Multicasting," IEEE/ACM Transactions on Networking, Vol. 6, No. 4, August 1998, pp. 461-474.

[41] Anees Shaikh, Kang G. Shin, "Destination-Driven Routing for Low-Cost Multicast," IEEE Journal on Selected Areas in Communications, Vol. 15, Issue: 3, April 1997, pp. 373-381.

[42] Xu Zhengquan, Chen Lin, "An Effective Heuristic Algorithm for Dynamic Multicast Routing with Delay-Constrained," 9th International Symposium on Computers and Communications 2004, Vol. 2, pp. 1024-1031.

[43] H. F. Salama, D. S. Reeves, Y. Viniotis, "Evaluation of Multicast Routing Algorithms for Real-time Communication on High Speed Networks," IEEE Journal on Selected Areas in Communications, Vol. 15, No. 3, pp. 332-34.

[44] Zbigniew Dziong, Ming Jia, L. G. MASON, "Analysis of Multicast Routing Algorithms for Broadband Networks," Proceedings of IEEE ATM Workshop, May 1998, pp. 186-194.

[45] T. H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Introduction to Algorithms, The MIT Press, McGraw Hill Book Company, 1998.

[46] A. Ballardie, "Core Based Tree (CBT) Multicast Routing Architecture," Internet RFC 2201, September 1997.

[47] Seoj Joo Koh, Shin Gak Kang, "Enhancement of CBT Multicast Routing Protocol," Proceedings of 8th International Conference of Parallel and Distributed Systems 2001 (ICPADS '01), June 2001, pp. 209-213.

[48] Ian Foster, Carl Kesselman, The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann Publishers, 1999.

[49] K. L. Calvert, M. B. Doar, E. W. Zegura, "Modeling Internet Topology," IEEE Communications Magazine, June 1997, Vol. 35, Issue: 6, June 1997, pp. 160-163.

[50] College of Computing, Georgia Institute of Technology, "Modeling Topology of Large Internetworks," http://www.cc.gatech.edu/projects/gtitm/.

[51] Amotz Bar-Noy, Sudipto Guha, Joseph Naor, Baruch Schieber, "Message Multicasting in Heterogeneous Networks," SIAM Journal of Computing, Vol. 30, No. 2, 2000, pp. 347-358.

[52] Eli Brosh, Yuval Shavitt, "Approximation and Heuristic Algorithms for Minimum Delay Application-Layer Multicast Trees," IEEE INFOCOMM 2004, March 2004.

[53] Mohammed Banikazemi, "IP Multicasting: Concepts, Algorithms, and Protocols IGMP, RPM, CBT, DVMRP, MOSPF, PIM, MBONE", Technical Report, Ohio State University, September 1997.

[54] H. Takahashi and A. Matsuyama, "An Approximate Solution for the Steiner Problem in Graphs," Mathematica Japonica, Vol. 24, No. 6, 1980, pp. 573-577.

[55] S. Ramanathan, "Multicast Tree Generation in Networks with Asymmetric Links," IEEE/ACM Transactions on Networking, Vol. 4, No. 4, 1996, pp. 573-568.

[56] L. Kou, G. Markowsky, and L. Berman, "A Fast Algorithm for Steiner Trees," Acta Informatica, Vol. 14, pp. 145-151.

[57] J. E. Hopcroft, R. M. Karp, "An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs," SIAM Journal of Computing, Vol. 2, 1973, pp. 225–231.

[58] O. Gross, "The Bottleneck Assignment Problem: An Algorithm," Proceedings of Rand Symposium on Mathematical Programming, Rand Publications R-351, Philip Wolfe, Editor, 1960, pp. 87-88.

[59] E. Lawler, Combinatorial Optimization: Networks and Matroids, Holt, Rheinhart and Wilson Publishing Company, 1976.

[60] A. Feder, R. Motwani, "Clique Partitions, Graph Compression and Speeding-up Algorithms," Journal of Computer System Sciences, Vol. 51, 1995, pp. 261–272.

[61] M. R. Garey, D. S. Johnson, Computers and Intractability, Freeman, San Francisco, 1979.

[62] Jeffrey Considine, John W. Byers, Ketan Mayer-Patel, "A Constraint Satisfaction Approach to Testbed Embedding Services," ACM SIGCOMM Computer Communications Review, Vol. 34, No. 1, January 2004, pp. 137-142.

[63] Shankar M Banik, Sridhar Radhakrishnan, Tao Zheng, Chandra N Sekharan, "Distributed Floor Control Protocols for Computer Collaborative Applications on Overlay Networks," 1st IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom 2005), San Jose, CA, December 2005.

[64] Shankar M Banik, Sridhar Radhakrishnan, Chandra N Sekharan, "Multicast Routing with Delay and Delay Variation Constraints for Collaborative Applications on Overlay Networks," IEEE Transactions on Parallel and Distributed Systems, 2006 (accepted).

[65] Gabriel Robins, Alexander Zelikovsky, "Improved Steiner Tree Approximation in Graphs," Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, CA, 2000, pp. 770-779.