ABSTRACT

HABERMAN, BRIAN KEITH. Cost, Delay, and Delay Variation Conscious Multicast Routing. (Under the direction of Professor George N. Rouskas.)

We study the problem of generating multicast trees to control the communication between members of a multicast group over a packet-switched network. These trees must support the quality-of-service requirements of real-time applications. The multicast tree must satisfy three objectives : (1) bounded delays between the source and all destinations, (2) minimized cost of the multicast tree, and (3) bounded variation among the delays along the paths to all destinations.

The primary contribution of this thesis is a novel strategy for the constrained Steiner tree problem. We present a heuristic that gives a good average case behavior for the maximum delay variation among the destinations as well as a reduction in the overall cost of the multicast tree. Our heuristic compares favorably with existing multicast algorithms in terms of tree cost, system running time, and the satisfaction of the delay and delay variation constraints.

COST, DELAY, AND DELAY VARIATION CONSCIOUS MULTICAST ROUTING

by

Brian K. Haberman

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Computer Science

Raleigh

1997

APPROVED BY:

Chair of Advisory Committee

BIOGRAPHY

Brian Haberman was born January 27, 1969, in Berea, Ohio. He received the Bachelor of Science degree in Computer Science from Clemson University in May, 1991. In 1995, he joined the Computer Science Department at North Carolina State University to pursue a Master of Science degree in Computer Science. He is employed in the Networking Division of IBM in the Research Triangle Park, NC.

ACKNOWLEDGEMENTS

I would like to take this opportunity to thank everyone who has helped me through this intellectual adventure.

First, I would like to express my sincere gratitude to my advisor, Professor George Rouskas. He has given me invaluable guidance, advice, and encouragement.

I would also like to thank the members of my advisory committee, Professors Doug Reeves and Harry Perros, for their time and helpful advice.

To Tom and Sally Parsons, I owe a big thank you for the support and encouragement they have given me throughout my pursuit of this degree.

To my parents, Jack and Phyllis, all the thanks in the world for the love and support they gave me.

And, of course, the biggest thanks go to my wife, Rebecca, and my daughter, Victoria. They were always there with love, understanding, and support when I needed it most. Without them, this work would have been impossible.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Multicast Communication

The recent advances in computer networking technology has led to the development of new network-centric applications. These applications have varying requirements for the underlying network. One of the main requirements is the ability to concurrently send messages to multiple users. Mechanisms to support *multicast* communication are becoming an important aspect in the design of distributed systems [20]. To fulfill this need, algorithms have been developed to manage the transmission of data from a user to multiple destinations. The primary method of implementing a reliable multicast communication algorithm has been through the use of a *multicast tree* [8].

It should be noted that the problem of generating multicast trees does not address the problem of reliable communication. The multicast routing algorithms do not have the responsibility of managing connection reliability. The higher-level transport protocols are responsible for maintaining the connection reliability. A survey of the transport layer protocols that address the problem of reliable mulitcast communication is presented in Section 2.5.

The growing interest in multicast communication has led to the development of several algorithms to generate multicast trees. These algorithms have been developed based on several different optimization goals. Shortest path algorithms attempt to

minimize the cost of the paths, or delay, from the source to each of the destinations. Several shortest path algorithms are presented by Bertsekas and Gallager [3]. Minimum Steiner tree algorithms attempt to minimize the total cost of the multicast tree. The minimum Steiner tree problem is $\mathcal{NP}$-complete, but several heuristics have been developed [8, 9, 23].

There are types of applications that have a requirement that the *variation* of delay must be bounded between the destinations. With video-conferencing now available on the Internet [12], it is desirable for all of the viewers to see and hear the speaker at the same time. In an on-line software distribution system, the primary host will want to guarantee that all the remote hosts get a new copy of software at the same time so that the sites never lose integrity. This constraint was first used as a performance measure in a study of existing multicast algorithms [17, 18].

Other methods of satisfying the variation of delay objective are possible. Buffering at the source, at the switching nodes, or at the receivers may be used. Buffering at the source would require the source to maintain additional information about all destinations. In addition, the use of a multicast tree for routing would no longer be needed since each message would have to be buffered a different amount of time for each destination. This causes the sender to incur the added cost of having to transmit the same message multiple times. Buffering at the switching nodes suffers from the same problem. The nodes must now add complexity in order to support the buffering function. Buffering at the receivers is a straightforward approach. The receiving protocol buffers the message for an amount of time equal to the delay variation. However, this requires the receivers to cooperate with the other members of the multicast group. In the software distribution example above, this approach is usable. In a scenario where the receivers are competing against one another, as in a distributed game, the network cannot rely on the receivers to delay messages. Additionally, the amount of buffering needed in this approach is directly proportional to the maximum variation of end-to-end delays. By providing bounds for the delay variation, more efficient usage of buffer space will result. We believe that buffering at the receivers may be used along with delay variation-based multicast routing algorithms to more successfully address the problems caused by variation in end-to-end delay.

The *delay variation* constraint is the basis of a new multicast algorithm, *Delay Variation Multicast Algorithm*(**DVMA**), presented by Rouskas and Baldine [2, 15]. Since the **DVMA** algorithm is concerned with the delays between sender and destinations, it does not take into account the total cost of the tree. The main contribution of our work is an extension of **DVMA** that takes into consideration the cost of the resulting multicast tree. We will generate a multicast tree that will meet the delay and delay variation constraints as well as minimize the cost of the multicast tree.

1.2 Organization

The rest of this thesis is organized as follows. Chapter 2 will present an overview of existing multicast algorithms. Chapter 3 will detail our method of modeling multicast communication over a packet-switched network. In Chapter 4 we will describe the problem of meeting all three of the optimization goals. Chapter 5 will describe the heuristic algorithms that we developed to meet the added cost objective. The test environment will be described in Chapter 7. The results obtained will be presented in Chapter 8 and our conclusions are in Chapter 9.

Chapter 2

Overview of Previous Work

The problem of reliable multicast communication has been studied extensively. The existing multicast tree algorithms can be categorized by their optimization goals. Historically, the two most common objectives have been the minimization of the end-to-end delay from source to each destination and the minimization of the overall cost of the multicast tree. Recently, a third constraint has been introduced : the minimization of the variation between the end-to-end delays [17, 18, 2]. The following sections will provide an overview of these constraints. The goal of the delay, or *shortest path*, constraint and algorithms developed for this constraint will be reviewed in Section 2.1. The least cost tree, or *Steiner tree*, algorithms will be described in Section 2.2. The algorithms that attempt to satisfy both objectives are called *constrained* Steiner tree algorithms and they are reviewed in Section 2.3. Section 2.4 will highlight the development of the delay variation constraint as well as the algorithm designed for it. Section 2.5 will survey several multicast transport protocols that can be used along with the multicast routing algorithms to guarantee reliable end-to-end communication.

2.1 Shortest Path Algorithms

The primary goal of the shortest path constraint is to minimize the time it takes to deliver information from a source to a destination. This constraint is the user's

view of the multicast connection. This problem can be solved, in polynomial time, by any of the existing shortest path algorithms. Dijkstra's shortest path algorithm [1] is one example. Dijkstra's algorithm creates a tree of shortest delay paths that span the source node and all the destination nodes. The network is represented by a graph where each edge represents the propagation delay between the nodes connected by that edge. If the user specified delay constraint cannot be satisfied by this tree of shortest paths, then no tree can satisfy the constraint.

2.2 Steiner Tree Algorithms

Section 2.1 shows how the problem of generating multicast trees can be solved to satisfy the user's network requirement. The second approach to generating multicast trees attempts to minimize the use of network resources. This problem corresponds to the Steiner tree problem in graph theory. If the network is represented by a graph, the edges in the graph represent a weight or cost, which is a measure of the utilization of that physical connection. The Steiner tree problem involves constructing a tree that spans a set of nodes and contains the minimum cost on its edges. This problem has been proven $\mathcal{NP}$-complete [6]. This means that a polynomial time solution does not exist for this problem, unless $\mathcal{P} = \mathcal{NP}$. Several heuristics do exist that can approximate the optimal solution.

Kou, Markowsky, and Berman developed the **KMB** [9] heuristic which has a worst-case time complexity of $\mathcal{O}(mn^2)$, where m is the size of the multicast group and n is the number of nodes in the graph. The **KMB** algorithm first constructs a complete undirected distance graph, G_1, containing all Steiner points[1] from the original graph G. A minimum spanning tree of G_1 is then found using Prim's or Dijkstra's minimum spanning tree algorithm. A new subgraph G_2 of G is formed by replacing each edge in the minimum spanning tree of G_1 by its corresponding shortest path in G. A new minimum spanning tree is generated from the G_2 subgraph. This tree is then pruned of any leaves that are not Steiner points.

[1] The Steiner points consist of the source node and all the destination nodes of a multicast group

Ramanathan presented a heuristic algorithm [14] which solves the Steiner tree problem for **directed** graphs. This algorithm builds a spanning tree in a step-wise method. The tree is initialized with the multicast source node as being the only element in the tree. At every step, a directed path is extended from the tree to a multicast destination node, thus adding the destination to the tree. This process is continued until all destination nodes have been added to the tree. The selection of which path to add is done based on a priority scheme. The nodes already in the tree are assigned a priority and put into a queue. The path that is chosen for inclusion in the tree is the path of least cost from a destination node to a member of the priority queue. The unique characteristic of this algorithm is the *control knob* feature that dictates to what degree the algorithm will be concerned about tree cost versus runtime efficiency.

2.3 Constrained Steiner Tree Algorithms

The constrained Steiner tree algorithms attempt to generate a minimum cost tree without violating the user imposed delay constraint. This approach requires the generation of a graph to represent the network. The difference is that each edge in the graph now has two costs associated with it. The first cost is the delay encountered while traversing the link. The second cost is the utilization of the link. Since the constrained Steiner tree problem contains the Steiner tree problem, it is also an $\mathcal{NP}$-complete problem. The only solutions for this problem are heuristics, such as **KPP**, **BSMA**, and **CAO**.

Kompella, Pasquale, and Polyzos developed the **KPP** algorithm [8] as a solution to the constrained Steiner tree problem. **KPP** assumes that the delay bound, Δ, is an integer. It then computes a constrained closure graph. If the delay bound assumption holds, this can be done in polynomial time. Once the closure graph is generated, a spanning tree is created from this graph by incrementally adding edges with minimum cost that do not violate Δ.

The *Bounded Shortest Multicast Algorithm*, **BSMA**, approach was developed by Zhu, Parsa, and Garcia-Luna-Aceves [23]. **BSMA** starts by calculating a shortest

path tree for the source and given multicast destination group. It then iteratively replaces paths in the tree with cheaper cost paths not in the tree that do not violate the delay bound until the total cost of the tree cannot be reduced any further. **BSMA** uses a *kth*-shortest paths algorithm to generate the set of replacement paths used in the path switching. The running time for **BSMA** is $\mathcal{O}(k|V|^3 log|V|)$, where k is the number of paths found by the *kth*-shortest paths algorithm and V is the set of nodes in the graph.

Widyono's solution to the constrained Steiner tree problem is the *Constrained Adaptive Ordering*, or **CAO** algorithm [22]. **CAO** uses a constrained Bellman-Ford algorithm to iteratively connect multicast group members to the source. The constrained Bellman-Ford algorithm is a breadth-first search to find the least cost path from the source to all other nodes in the network. After each iteration of the Bellman-Ford algorithm, the destination node with the cheapest path to the source is added to the existing tree. It was found, however, that there are cases in which the running time of **CAO** grows exponentially with respect to the number of nodes in the graph [18].

2.4 Delay Variation Constrained Algorithms

The *delay variation* constraint is the latest user requirement for multicast trees. It was first used as a performance metric for a comparative study of multicast routing algorithms [17, 18]. The need for minimizing the delay variation between destinations can be seen as reducing the competitive advantage a destination gains by being able to process a message sooner than another destination.

The first use of delay variation as a user constraint in a multicast tree algorithm wass presented by Rouskas and Baldine [2, 15]. The *Delay Variation Multicast Algorithm*, or **DVMA**, operates under two user imposed constraints, the delay bound, Δ, and the inter-destination delay variation, δ. **DVMA** will return a multicast tree that either meets the user imposed constraints or it returns a tree that meets Δ and has the least value of δ.

DVMA initially constructs a tree T_0 of shortest paths from s to all destination

nodes using Dijkstra's algorithm [4]. If T_0 does not satisfy Δ, then no tree can satisfy it and the algorithm terminates. The value of Δ will then need to be re-negotiated. If T_0 satisfies Δ and δ, then T_0 is the solution for the specified network and is used for the multicast session. T_0 is the optimal solution for the delay bound constraint, Δ.

If T_0 satisfies Δ, but does not satisfy δ, **DVMA** begins a heuristic search to construct a new tree which satisfies both Δ and δ. **DVMA** does this by employing an approach similar to that used by **BSMA**. **DVMA** employs incremental path switching to generate a tree T that meets both Δ and δ. The algorithm maintains a list of destination nodes not in the tree. It then picks a destination node from this list and finds the k shortest paths from it to a node in the tree. These k paths must not use any edges or nodes already in the tree T. The k shortest paths are chosen so that if no feasible tree can be found, the total delay along the path will be at most Δ, and the tree resulting from the addition of this path will have the lowest delay variation among the possible trees constructed by adding the destination node using other paths.

The complexity of **DVMA** is dominated by the path switching. In the worst case, **DVMA** has a complexity of $\mathcal{O}(klmn^4)$, where k is the number of paths generated from the source to a non-attached destination node, l is the number of paths from a non-attached destination to an attached node, m is the number of destination nodes, and n is the total number of nodes in the network.

2.5 Reliable Multicast Transport Protocols

The use of multicast routing protocols does not guarantee reliable communication between the sender and the destination group. That responsibility falls on the transport layer protocol. The need for reliable multicast communication increases as more collaborative, interactive applications become available. In our video-conference example, the sender would like a guarantee that data will be delivered in an orderly manner to ensure consistent presentations across the destination group. In the case of the software distribution system, the sender wants to guarantee *reliability* of the

data being delivered to avoid having sites in an inconsistent state. In either case, the multicast transport protocol controls the reliability of the communication sessions. There are several multicast transport protocols that have been proposed and/or implemented. Section 2.5.1 describes a transport protocol based on a multicast transmission with negative acknowledgements mechanism. In Section 2.5.2, we survey a multicast transport protocol that guarantees *complete reliability* by using a hierarchical approach.

2.5.1 Reliable Adaptive Multicast Protocol

The *Reliable Adaptive Multicast Protocol* (**RAMP**) was described by Koifman and Zabele [7]. **RAMP** is designed to guarantee reliable, orderly delivery of multicast packets. The protocols foundation is based on the underlying network on which it was developed. The ARPA-sponsored **T**est**B**ed for **O**ptical **NE**tworking (**TBONE**) is an all-optical, circuit switched, gigabit network. However, the packet loss characteristics of **TBONE** resemble those of switched virtual circuit ATM networks and packet-switched networks that employ reservation services. This means that **RAMP**'s design is functional for the next generation of packet switched networks.

RAMP is a transport layer multicast protocol designed to operate over network layer multicast protocols such as IP multicast. Unlike some multicast transport protocols, **RAMP** is *fully reliable*. This means that an application is notified whenever a receiver or a sender fails. For collaborative interactive applications in which each node is a sender and a receiver to a multicast group, this is highly desirable. And because **RAMP** maintains state information about all members of the multicast group in order to guarantee full reliability, source knowledge of the destination set is known, thus eliminating the need for Internet Group Management Protocol services.

The **TBONE** network has extremely low bit-error rates (10^{-12} or better) and the switches do not have any store-and-forward capability. Because of these characteristics, packet loss in **TBONE** is almost always caused by buffer overflow at the receiver. **RAMP** exploits this by using a receiver-initiated, *NAK*-based, unicast error notification mechanism. When the receiver detects a packet loss, it immediately

transmits a unicast *NAK* packet to the sender. The sender then re-transmits the lost packet as a unicast packet to the receiver that initiated the *NAK*. This method of flow control is based on the fact that receiver losses are independent. If the lost packet is retransmitted as a multicast packet, the other receivers are forced to unnecessarily process and discard the packet.

The performance of **RAMP** is mixed. Over **TBONE**, the protocol performs well when compared against TCP and UDP. The observed throughput compares well with the theoretical values. Packet loss measures and recovery tests show that **RAMP** performs very well. However, its performance suffers when applied to other network topologies. Test-cases run over Ethernet show that duplicate transmission handling and flow control performance degrades faster than can be justified. Additional developmental work is being done on **RAMP** [7].

2.5.2 Reliable Multicast Transport Protocol

For some applications, such as a software distribution system or medical imaging systems, data reliability is more important than bounded delay. The *Reliable Multicast Transport Protocol* (**RMTP**) [11] is designed to guarantee *complete reliability*. **RMTP** provides sequenced, lossless delivery of bulk data from a sender to a multicast group.

The basic ideas behind **RMTP** were derived from the *Designated Status Protocol* (**DSP**) [13]. In particular, **RMTP** employs a hierarchical approach to message acknowledgements. The use of *Designated Receivers* helps to avoid the *ACK*-implosion problem. This allows for an efficient selective retransmission scheme that does not flood a sender. Any retransmission requests are first handled by the receiver's designated receiver. The designated receiver for an area caches all transmissions by the sender and responds to retransmission requests with the data stored in the cache.

RMTP employs a windowed flow control mechanism with congestion avoidance. It makes no assumptions about the buffer space or processing power at any of the receivers. The protocol also allows for receivers to *throttle back* the sender during a congestion period. Because the communication is multicast, the sender will only send

data to the group as fast as the slowest receiver can handle it. While this degrades performance for other receivers, it maintains the data reliability for all receivers.

Currently, **RMTP** is still in a prototype stage. The preliminary testing shows promise. The use of the designated receivers keeps the sender from being flooded with retransmission requests. The protocol also adapts well to receivers in various network environments. However, it does this at the cost of performance on faster networks. Additional details into the implementation of **RMTP** are outlined by Lin and Paul [11].

Chapter 3

Network Model

We consider the problem of generating communication paths through a packet-switched network for multicast traffic. We model the network as a simple[1], connected, directed graph $G = (V, E)$, where V denotes the set of nodes in the network, and E represents the set of edges. These edges correspond to the communication links between the nodes in the network. The existence of a link $e = (u, v)$ from u to v implies the existence of a link $e' = (v, u)$ for any $u, v \in V$.

Any edge $e \in E$ will have two associated weights. The *link delay function* $\mathcal{D}(e)$ assigns a non-negative weight to each edge. $\mathcal{D}(e)$ represents the delay a data packet experiences on that link, and it is the sum of the switching, queueing, transmission, and propagation delays. The *link cost function* $\mathcal{C}(e)$ is a function of the amount of traffic traversing link e and the amount of buffer space needed for that traffic. In other words, it is a measure of the link utilization. Because the network is asymmetric, it is often the case that $\mathcal{C}(e) \neq \mathcal{C}(e')$ and $\mathcal{D}(e) \neq \mathcal{D}(e')$.

In the multicast communication scenario, a *source node* $s \in V$ is transmitting data packets destined for a *multicast group* $M \subseteq V$, where $\mid M \mid \leq \mid V \mid$. In order for multicast communication to proceed, *multicast connections* are established. During this establishment period, the source node s must determine paths to all members of M. These paths are used to construct a *multicast tree* $T = (V_T, E_T)$, $V_T \subseteq V$ and $E_T \subseteq E$, rooted at s. Nodes which are not members of M may be contained in V_T

[1] The graph will contain no more than one edge between an ordered pair of nodes

as relay nodes. Relay nodes forward packets along adjacent links without actively participating in the multicast communication.

We define $P_T(s, g) \subseteq E_T$ to be the set of links from s to $g \in M$. The total delay experienced along this path is $\sum_{e \in P_T(s,g)} \mathcal{D}(e)$. The cost of $P_T(s, g)$ is $\sum_{e \in P_T(s,g)} \mathcal{C}(e)$. The application utilizing the multicasting services must supply two parameters that specify the *quality of service*, **QoS**, required. The *delay tolerance*, Δ, represents an upper bound on the acceptable end-to-end delay along the path from s to any $g \in M$. The maximum acceptable difference between the end-to-end delays along any two paths $P_T(s, g)$ and $P_T(s, f)$, $g, f \in M$, is represented by the parameter δ. These parameters specify the requirements for the application to run. If these parameters cannot be met, the application will abort. In addition, the cost of the multicast tree must be the minimal cost tree that meets the user constraints. Therefore, we will now discuss the problem of balancing the user requirements with the network requirement.

Chapter 4

Steiner Trees with Delay and Delay Variation Constraints

With the network model described in Chapter 3, we can now formally define the problem of generating a multicast tree that meets the delay and delay variation constraints with minimal cost. We allow Δ to be the *delay* constraint and δ to be the *delay variation* constraint as specified by the user application attempting to initiate a multicast session. The *Delay and Delay Variation-Bounded Multicast Tree*(DVBMT) problem was first identified by Baldine [2]. The problem we address in this work integrates DVBMT with the problem of minimizing the overall cost of the multicast tree. The objective is to construct a multicast tree such that the delays along the paths from the source to the destinations are within the user constraints and the overall cost of the tree is minimized. More formally, we express the *Cost Conscious Delay and Delay Variation-Bounded Multicast Tree* problem as follows.

Problem 1 *(***CCDVBMT***) Given a network $G = (V, E)$, a multicast group $M \subseteq V$, a source node $s \in V$, a link delay function $\mathcal{D}$, a link cost function $\mathcal{C}$, a delay constraint Δ, a delay variation constraint δ, and an overall cost C, does there exist a tree $T = (V_T, E_T)$ spanning s and all the nodes in M, such that :*

$$\sum_{e \in P_T(s,g)} \mathcal{D}(e) \leq \Delta \quad \forall g \in M \tag{4.1}$$

$$\left| \sum_{e \in P_T(s,g)} \mathcal{D}(e) \; - \sum_{e \in P_T(s,f)} \mathcal{D}(e) \; \right| \leq \delta \quad \forall g, f \in M \tag{4.2}$$

$$\sum_{e \in E_T} \mathcal{C}(e) \; \leq \; \mathsf{C} \tag{4.3}$$

If a set of trees can be found that satisfies both (4.1) and (4.2), the tree of least cost will be chosen as a solution to the problem. In the event that only a single tree is found that satisfies (4.1) and (4.2), it is the solution to the problem, regardless of its overall cost. When no tree can be found that meets the Δ and δ constraints, the multicast session cannot continue. More formally, in order for the multicast session to proceed, it is necessary and sufficient that (4.1) and (4.2) be met by at least one tree.

The CCDVBMT problem can be seen to be $\mathcal{NP}$-complete. If the constraints (4.1) and (4.2) are removed, the problem reduces to the Steiner tree problem. The Steiner tree problem is a well-known $\mathcal{NP}$-complete problem [5]. CCDVBMT also reduces to the DVBMT problem [2, 15] if the constraint (4.3) is removed. The DVBMT problem was also proven to be $\mathcal{NP}$-complete [2, 15].

An observation made is that (4.1) and (4.2) are conflicting objectives and that a balance must be struck between the two constraints [2]. In essence, the longest delay path in G will affect the selection of all other paths. With the addition of a cost component, a balance must now be found between three conflicting requirements.

The delay constraint (4.1) has been considered in several previous works on designing multicast trees [1, 8, 23]. The cost objective (4.3) is the basis for several algorithms that minimize the network resources [9, 14]. Algorithms that balance these two constraints address the *constrained Steiner tree* problem and several representative examples are available [8, 22, 23]. The delay variation constraint (4.2) was first introduced as a performance metric [17, 18]. Until the introduction of **DVMA** [2], the delay variation constraint had not been considered in the construction of multicast trees. To the best of our knowledge, this work is the first that takes all three objectives into account.

Chapter 5

Multicast Tree Algorithms for CCDVBMT

Chapter 4 formulated the CCDVBMT problem, and now we will attempt to solve it. Because the problem in question is $\mathcal{NP}$-complete, a polynomial time solution only exists if $\mathcal{P} = \mathcal{NP}$. In this chapter, we will introduce several heuristics to solve to the *CCDVBMT* problem.

5.1 Algorithm Description

We make the assumption that the node initiating the multicast session has complete knowledge of the network topology. There are several protocols available that can be used to collect this information [3]. Once this information is collected, the algorithm performs the following steps.

1. Generate a tree T_0 of least cost paths to each destination
2. Run the **CCDVMA** algorithm to obtain a new tree T
3. If T satisfies (4.1) and (4.2), return T and stop
4. If T does not satisfy (4.2), re-negotiate δ
5. If T satisfies the new δ, return T and stop

6. Stop

The main difference between this algorithm and the one presented by Rouskas and Baldine [2, 15], is that we make the assumption that the delay bound Δ and delay variation bound δ are inter-related. That is, the final delay variation value for any tree T is going to have an affect on the delays along all paths in T. For that reason, we do not test the delay bound individually for compliance until the invocation of the **CCDVMA** algorithm.

The algorithm generates a tree T_0 of least cost paths using Dijkstra's algorithm [4]. The algorithm then performs a search algorithm in an attempt to construct a tree that will satisfy (4.1) and (4.2). It should be noted that it is possible that a feasible tree does not exist for the CCDVBMT problem. In this situation, the application has the choice of abandoning the connection or re-negotiating the values of Δ and δ. If the application can find an acceptable value for the constraints, the source node would go through another iteration of the steps outlined above.

In order to facilitate the re-negotiation process, the search algorithm has been designed to return the tree which comes closest to meeting the δ constraint if a feasible tree is not found. If two or more trees exist with the same δ, the tree of least cost is returned. So, even though a solution to the CCDVBMT problem may not exist, the best tree that can be obtained, given the network topology, is available to the application. If this tree meets the re-negotiated constraints, the application does not have to re-run the algorithm.

The following sections present two new multicast tree heuristics designed to solve the CCDVBMT problem. Section 5.2 describes the **CCDVMA** algorithm and its relationship to the **DVMA** predecessor. Section 5.3 presents **CC2DVMA**, a modified version of **DVMA**.

5.2 Cost Conscious Delay Variation Multicast Algorithm (CCDVMA)

This section introduces the *Cost Conscious Delay Variation Multicast Routing Algorithm*(**CCDVMA**). The basic philosophy of **CCDVMA** is to start with a tree of smallest cost paths and incrementally replace paths in the tree in order to meet the delay and delay variation constraints. **CCDVMA** will always return a tree to the application. The tree returned will either be the tree of least cost found that satisfies Δ and δ, or it will be a tree of least cost found that comes closest to meeting both user constraints.

As stated above, we start with a tree of smallest cost paths T_0 spanning s and M. We can generate such a tree in $\mathcal{O}(n^2)$ using Dijkstra's algorithm [4] allowing the link cost function to be the utilization of that link rather than the delay measure. By generating the initial tree in this manner, we are starting with the optimal cost paths from the source to all the destinations. The search algorithm in Figure 5.1 is run to generate a set of candidate trees from which the best one is chosen.

The search algorithm modifies the tree T_0 using a form of path substitution. We apply a similar logic as that used in **DVMA** [2, 15]. We assume that a tree $T = (V_T, E_T)$ spanning s and a subset of M has been determined. We let $U = M - (M \bigcap V_T)$ be the set of destinations not yet in T. We then incrementally build T using the following three steps:

1. Select a destination node $u \in U$
2. Find a good path from u to $v \in V_T$
3. Construct a new tree T' by connecting all nodes and edges in this path to T, and update U to exclude u and any other node used in the path

In the **DVMA** approach, step 2 chose paths that did not use any nodes already in the tree other than v or any edges in E_T. Our approach is to allow as much *path sharing* as possible. Path sharing is a crucial element in the reduction of the overall tree cost [8]. The selection of a path in step 2 should allow the resulting tree in step 3

```
M is the set of destinations
T_0 is the tree of smallest cost for graph G = (V, A)
BEGIN
1   T = T_0
2   Find the first k paths, p_1..p_k, of smallest cost from the source
    s to a destination d ∈ M, where d is the node with the
    highest cost path that does not exceed Δ
3   for i =  1 to k do
4       Initialize T_i = (V_i, A_i) to include path p_i
5       Let U = M − (M ∩ V_i) be the set of destinations
        not yet connected to the tree T_i
6       while U ≠ ø do
7           Pick any node u ∈ U
8           for each node v ∈ V_i do
9               Construct a new graph G' from G by excluding all nodes
                in V_i − {v} and all links in A_i
10              Find the first l least cost paths from v to u
                in the new graph G'
11              Of these l paths, choose the least delay path and call it q_v
12          end of for loop
13          Select the best path q among all paths q_v, v ∈ V_i
14          Update T_i = (V_i, A_i) to include all nodes and links in path q
15          Update U to remove links and nodes now in T_i
16      end of while loop
17      Let T be the tree among T and T_i with the smallest cost that
        satisfies δ
18  end of for loop
19  return T
END
```

Figure 5.1: CCDVMA algorithm for the CCDVBMT problem

to be a *feasible* tree for the subset of M it contains. In order to find that path, we generate a set of l paths of smallest cost from u to v. These paths are found using a graph G' created by excluding all nodes in V_i except v and all edges in E_i from the original graph G. This is done so that the addition of any of the l paths into T will not create a cycle. In order to handle the situation where there is no $feasible$ tree including a path from v to u, we repeat the process for u and all nodes $v \in V_T$. This allows the "good path" for u to be selected from all possible paths into T.

The above description assumes that we have a tree that spans s and a subset of M. We must address how we choose the initial subset. We start with a subset of size one, and pick the destination node that has the highest *cost* path in T_0 that does not have a delay component that exceeds Δ to be the initial member. The **DVMA** approach was to pick the node with the longest delay path because a shorter delay path did not exist. However, **CCDVMA** is working with an initial tree that has minimal cost not minimal delay. From this initial tree T, we then incrementally add destination nodes to the tree using the steps outlined above.

It should be noted that a multicast tree will be returned in all cases. This is based on the assumption that the graph G representing the network is connected. This means that there will always be *at least* one path from s to any node $g \in M$. The tree returned, however, may not be a *feasible* tree if there is no path in G from s to g that does not meet (4.1). In this situation, there is a conflict between the network and the user constraint Δ. This will cause an *infeasible* tree to be returned, but this tree is the best that can be done with the current network characteristics.

5.3 Cost Conscious 2 Delay Variation Multicast Algorithm ($CC2DVMA$)

The second algorithm designed for the CCDVBMT problem is called *Cost Conscious 2 Delay Variation Multicast Algorithm.* **CC2DVMA** is modeled more like the original **DVMA**. The algorithm logic is outlined in Figure 5.2. The two main differences are in the checking that is done on the initial tree and the decision making

process as to which tree is the **best** feasible tree. We will now describe these two differences in greater detail.

Dijkstra's shortest path algorithm is employed by **DVMA** to generate an initial tree of smallest delays. This tree T_0 is then tested to see if it satisfies (4.1) and (4.2). If it does, all the user constraints are satisfied, and the multicast session establishment can continue to the next phase. If (4.1) is satisfied but (4.2) is not, the **DVMA** search algorithm is applied to T_0. In **CC2DVMA**, the initial tree T_0 is generated using Dijkstra's shortest path algorithm as well. Then T_0 is passed directly to the **CC2DVMA** search algorithm for processing. We do not test the initial tree for compliance with (4.2) since we want to try and minimize the tree cost. How that is done is explained below.

If the initial tree did not meet the delay variation constraint, the **DVMA** search algorithm picked the tree T with the smallest δ_T. With **CC2DVMA** the tree with the lowest cost that satisfies the delay and delay variation constraint is chosen as the best feasible tree. This additional restriction forces the search algorithm to process all possible paths. This added cost causes the average-case complexity of **CC2DVMA** to be the same as its worst-case complexity.

Even though they differ in their approach, **CCDVMA** and **CC2DVMA** give very similar results. This will be seen in Section 8.

5.4 Complexity Analysis

The running time complexity of **CCDVMA** is directly derived from its predecessor **DVMA**. In our analysis

- k is the number of paths generated at line 2 of Figure 5.1
- l is the number of paths generated at line 10 of Figure 5.1
- $m =\mid M \mid$, the size of the multicast group
- $n =\mid V \mid$, the number of nodes in the network

```
M is the set of destinations
T_0 is the tree of least delay for graph G = (V, A)
BEGIN
1   T = T_0
2   Find the first k paths, p_1, .., p_k, of shortest
    delay from the source s to a destination d in M
    where the path to d is the highest delay path in T
3   for i =  1 to k do
4       Initialize T_i = (V_i, A_i) to include path p_i
5       Let U = M − (M ∩ V_i) be the set of destinations
        not yet connected to the tree T_i
6       while U ≠ ø do
7           Pick any node u ∈ U
8           for each node v ∈ V_i do
9               Construct a new graph G' from G by excluding all nodes
                in V_i − {v} and all links in A_i
10              Find the first l shortest paths from v to u
                in the new graph G'
11              Of these l paths, choose the least delay path and call it q_v
12          end of for loop
13          Select the best path q among all paths q_v, v ∈ V_i
14          Update T_i = (V_i, A_i) to include all nodes and links in path q
15          Update U to remove links and nodes now in T_i
16      end of while loop
17      Let T be the tree among T and T_i with the smaller
        cost and meets the δ_T constraint
18  end of for loop
19  return T
END
```

Figure 5.2: CC2DVMA algorithm for the CCDVBMT problem

The running time of **CCDVMA** is dominated by the loop from line 3 to line 18 in Figure 5.1. This loop is executed at most k times. During an iteration of this outer loop, the loop from line 6 to 16 is executed at most $m-1$ times. Inside this loop, the computation time is determined by the number of nodes in V_T and the l-shortest path algorithm at line 10. The algorithm at line 10 takes $\mathcal{O}(ln^3)$ [10]. The complexity of the innermost loop becomes $\mathcal{O}(ln^4)$. Therefore, the overall complexity of **CCDVMA** is $\mathcal{O}(klmn^4)$.

Rouskas and Baldine stated that the maximum value that parameters k and l could take is equal to the maximum number of paths of delay at most Δ, and they expected the maximum value of k and l to actually be small constants [15]. In addition, for our simulations and the simulations carried out by Rouskas and Baldine, k has an imposed upper bound of $3n$ and l has an upper bound of n [15, 2]. Measurements taken in our simulations show that k and l take on values much lower than the imposed upper limits. Appendix A show the measurements of k and l.

It also holds that **CC2DVMA** has a running time complexity of $\mathcal{O}(klmn^4)$. However, **CC2DVMA**'s average time complexity is the same as its worst-case time complexity. This is due to the fact that, unlike **DVMA**, **CC2DVMA** inspects every tree that it can generate. In **DVMA**, if the search algorithm is invoked, it terminates upon finding the first tree that satisfies (4.2). **CC2DVMA**, on the other hand, must inspect all of the possible trees in order to find the one of least cost that satisfies the user constraints.

Chapter 6

Dynamic Reorganization of the Multicast Tree

Chapter 5 presented the *static* implementation of the **CCDVMA** algorithm. All group members, $\forall g \in M$, were known prior to the invocation of **CCDVMA**. Now we address the question of *dynamic* updates to the multicast group.

The ability of a node to dynamically join or leave a multicast group is becoming more important. Applications, such as video-conferencing, do not expect the multicast group membership to remain constant through the life of the multicast connection. For that reason, we address the problems associated with any node $v \in G$ issuing a $LEAVE(M)$ or $JOIN(M)$ request at any time. After v makes such a request, we now have a new destination set, $M' = M - v$ or $M' = M \bigcup \{v\}$. However, we want to avoid having to run **CCDVMA** again to obtain a new tree T' for M'. By not re-running **CCDVMA**, we not only save on computation time, but we also avoid the expenses occurred in

- Tearing down existing multicast connections
- Constructing new connections
- Completing other steps in the connection establishment phase

Such an approach may also lead to other destination nodes experiencing interruption

in their service. For these reasons, we now present methods to allow dynamic multicast group updates with minimal cost in computation and the least amount of service interruption. We address dynamic $LEAVE(M)$ requests in Section 6.1 and dynamic $JOIN(M)$ requests in Section 6.2.

6.1 Leave **Requests**

When a node $v \in M$ issues a $LEAVE(M)$ request, there are two possible situations that need to be handled. Node v can either be a leaf node of the tree T, or it is an internal node of T. Neither situation results in interruption of the multicast service for nodes other than v.

If v is an internal node in the tree T, then the handling is simple. The multicast traffic is no longer forwarded to the user process at v. Node v remains a member of the tree T, but now is strictly a relay node for all members of M that are downstream of v. The resulting tree T' is identical to the original tree T and no node other than v has an interruption of service.

On the other hand, if v is a leaf node of T, we must prune T of v so that we do not waste bandwidth. This is accomplished by removing v and all relay nodes that only provide service for v from the tree T. This can be accomplished by traversing the path from v to s until we arrive at s or a node that has a downstream path other than the path to v. The pruning of T does not cause any interruption of multicast service for any node other than v.

6.2 Join **Requests**

In the case where node $v \notin M$ requests to join M, we must consider three different scenarios. Each scenario requires a different method for handling the request. We will explain each method and the affect it has on the existing multicast connection.

In the first situation, $v \notin V_T$ issues a $JOIN(M)$ request. The tree T must be augmented to include a path from a node $u \in V_T$ to the new node v. This can be accomplished by using **CCDVMA** (Figure 5.1) and initializing $T = (V_T, E_T)$ at line

4 and $U = \{u\}$ at line 5. We then execute lines 6 through 16 to search for a *feasible* tree for $M \bigcup \{u\}$. The addition of a path to u only involves u and does not cause an interruption of multicast service to any existing member of M. The time complexity of this method of tree augmentation can be seen to be $\mathcal{O}(ln^4)$.

If a *feasible* path cannot be found for u, there are two options. The first option is to tear down the multicast connections and invoke **CCDVMA** for the multicast group $M' = M \bigcup \{u\}$. The second, and less expensive, option is to deny admission to the group.

The second situation involves $v \in V_T$ issuing a $JOIN(M)$ request. If the constraint (4.2) is satisfied, v may begin to forward the multicast traffic to its user process. Since v is already in T, we know it is not a leaf node and that it satisfies (4.1). No interruption of the multicast session occurs and T is a feasible tree for the new multicast group M'.

The last case involves a $JOIN(M)$ request from a node $v \in V_T$. However, in this case, (4.2) is not satisfied. In order for v to become a member of M, a longer path from s to v must be found. In addition, all current nodes that are in the subtree rooted at v will be affected by this change. Consequently, we must re-calculate the paths from s to all destination nodes that are in v's subtree. Let $W \subset M$ be the set of nodes that are downstream of v. We again apply part of **CCDVMA** (Figure 5.1). Let T' be the tree created from T by excluding the subtree rooted at v. We allow $T = T'$ at line 4 and $U = W \bigcup \{u\}$ at line 5. The lines 6 through 16 are executed to create a new *feasible* tree for the new multicast group M'. The time complexity of this operation is $\mathcal{O}(lmn^4)$.

Chapter 7

Simulation Environment

7.1 Simulation Tool

The results from this research were obtained using the multicast routing simulator **MCRSIM©**[1], developed at North Carolina State University by Hussein Salama [16]. The **MCRSIM©** package allows the user to create network graphs, run multicast algorithms on these graphs, and simulate traffic flow.

The **MCRSIM©** package simulates actual ATM networks with fixed cell sizes and heterogeneous link capacities. It does the simulation at the cell level rather than the call level. This allows us to get actual end-to-end delay, delay jitter, and cell loss rates. A call level simulator can only estimate these measures.

MCRSIM© also supports multiple traffic types in order to simulate an actual high-speed network. The multicast source traffic can be voice or video. Both of these are variable bit rate sources suitable for a host of multimedia applications. The simulation package also models background traffic on each link. The background traffic can either be multiplexed video sources or an aggregate batch source.

There are several multicast routing algorithms implemented in **MCRSIM©**. The set of algorithms implemented allows for a meaningful comparison between our new algorithms and existing ones. The algorithms supported in this package are:

[1]MCRSIM can be downloaded from ftp://ftp.csc.ncsu.edu/pub/rtcomm/mcrsim.html

- **Unconstrained Algorithms** : KMP, Least Delay, Dijkstra's Least Cost, Bellman-Ford Least Cost, Minimum Spanning Tree, Reverse Path Multicasting, Water's SC heuristic, and a modified Water's SC heuristic
- **Delay Constrained Algorithms** : KPP, CAO, BSMA, and the optimal constrained Steiner tree algorithm

7.2 Simulation Description

The experiments were carried out using the random graph generator and random multicast group generator functions of **MCRSIM©**. The simulator implements a modified version of the random graph generator described by Waxman [21][2]. The nodes in the network have an average node degree of 4. The delay constraint Δ is set to 0.025 seconds which is an upper bound on the propagation delay across the network. The delay variation constraint is initialized to 0.01 seconds. The simulations were run using 3 variables:

1. Network Size : 40, 60, 80, and 100 nodes
2. Multicast Group Size : 5% and 10% of total network size
3. Traffic Load : 15% and 50% network utilization

Each permutation of the variables was simulated 100 times. The results of each run were averaged together to generate a composite score. If a run failed to generate a multicast tree, its data was not included in the averages. The reason being that if a multicast tree satisfying the parameters Δ and δ cannot be found, the user application will abort and not attempt to continue the multicast session.

The simulations measured the performance of the **CCDVMA** and **CC2DVMA** algorithms as well as the **DVMA**, **KPP**, and **BSMA** algorithms[3]. The **DVMA** algorithm was included since it is the forerunner of delay variation algorithms. **KPP**

[2]The modifications made guarantee the resulting network is connected and the minimum degree of any node is 2.

[3]Descriptions of the **DVMA**, **KPP**, and **BSMA** can be found in Section 2

and **BSMA** were chosen as representative heuristics for the constrained Steiner tree problem.

The five algorithms were compared using five measures.

1. Average path delay
2. Delay variation
3. Multicast tree cost
4. Percentage of simulations failing to generate a multicast tree
5. Total System Running Time

The fourth measure was included to determine the ability of an algorithm to create a suitable tree in relation to the other algorithms. It is also important since the tree measurements are not included in the averages when an algorithm fails to meet the user constraints.

The *Total System Running Time* measure was included to do a numerical comparison of the actual system running time for each algorithm. These measures were taken only for the actual time spent in the algorithm searching for a *feasible* multicast tree.

Chapter 8

Simulation Results

The results obtained are the *average case* behavior characteristics of the five algorithms. We have broken up the presentation of data into sections corresponding to the comparison measures described in Section 7. Section 8.1 presents the *average path delays* for our simulations. The *delay variation* comparison is in Section 8.2. Overall *tree cost* results are in Section 8.3. The ability of the five algorithms to *generate feasible trees* is presented in 8.4. Section 8.5 shows the comparison of *system running time* for the algorithms. And finally, Section 8.6 will combine these results into some general observations.

8.1 Average Path Delays

The average path delay is calculated by averaging the delay values of the paths from s to all members of the destination group M. If a multicast tree is generated, this average will always be less than Δ. Otherwise, at least one path in the multicast tree must be greater than Δ, and this condition would cause the tree to be an *infeasible* solution to the problem.

Figures 8.1 and 8.2 show the experimental results obtained for multicast groups of size 5% and 10%, respectively. It can be seen from these figures that the entire *DVMA* family of algorithms give a better average delay than the **KPP** and **BSMA** algorithms. Since the *DVMA*-based algorithms explicitly attempt to satisfy (4.2), the

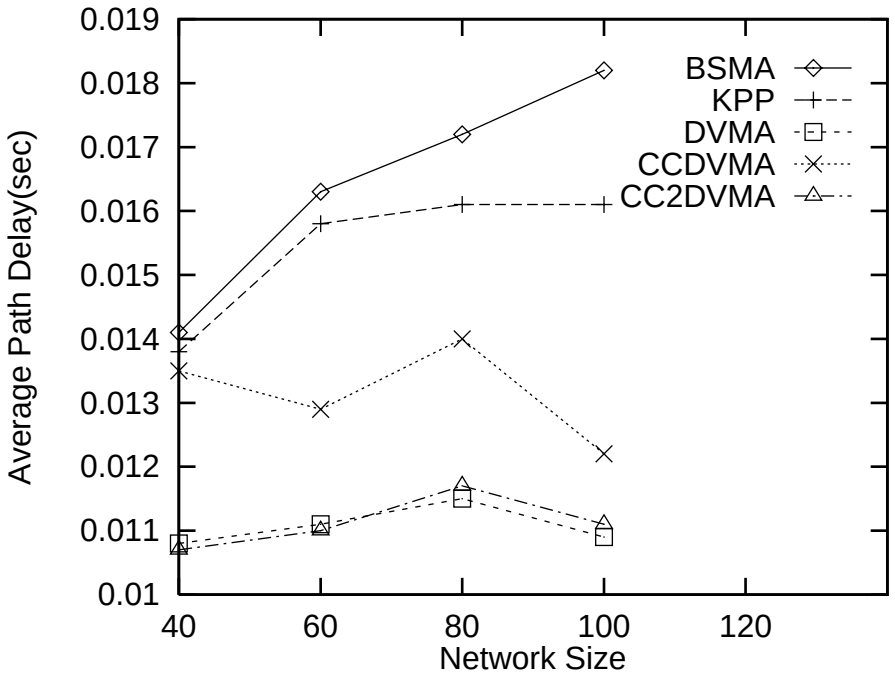

Figure 8.1: Average Path Delay for 5% Multicast Group membership and 15% network utilization

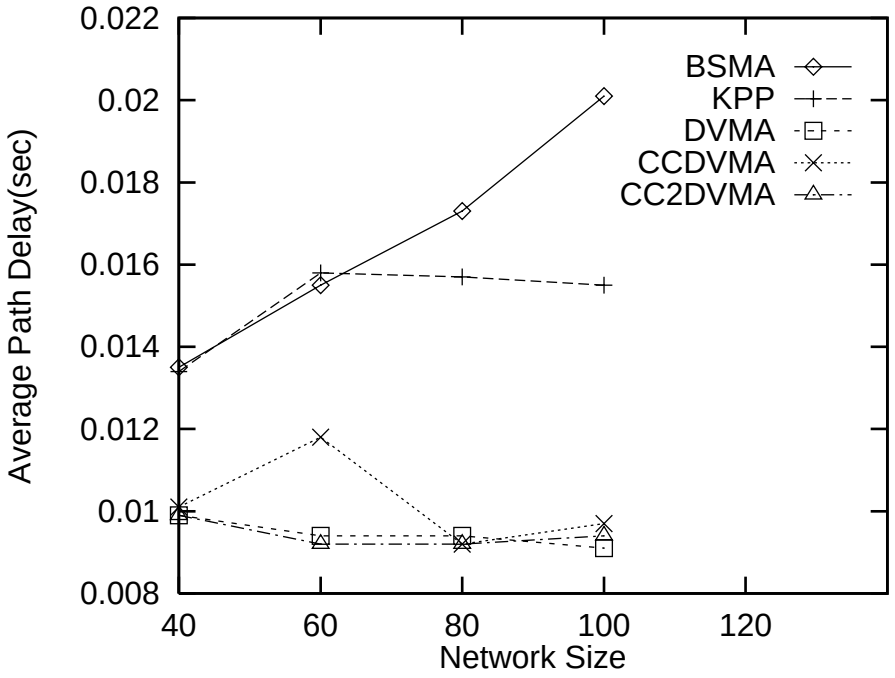

Figure 8.2: Average Path Delay for 10% Multicast Group membership and 15% network utilization

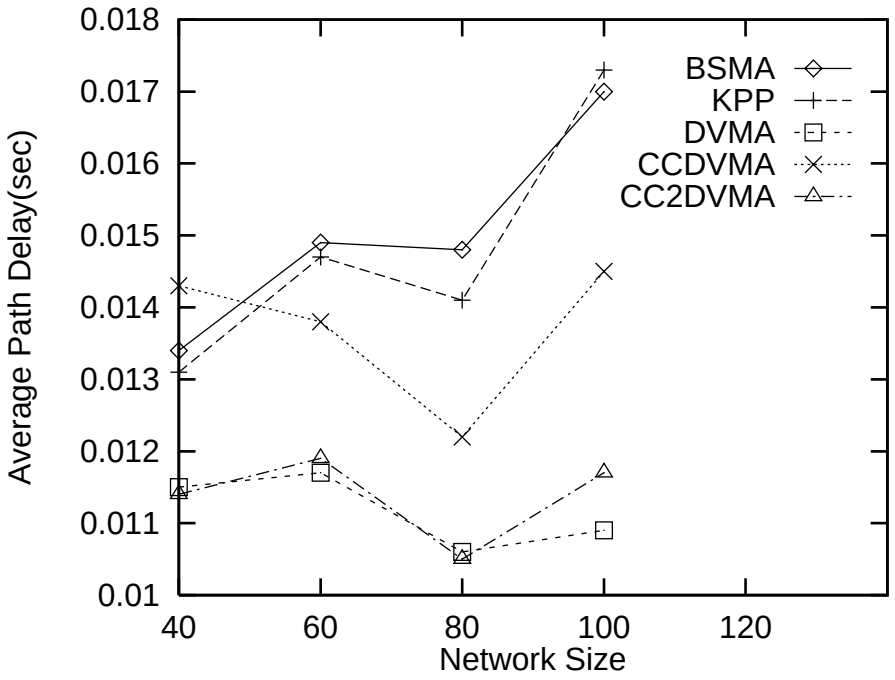

Figure 8.3: Average Path Delay for 5% Multicast Group membership and 50% network utilization

paths are chosen so that their delay factors fall within δ. This means that the paths chosen will have a tight average delay with a small statistical variance.

In Figure 8.3, the network traffic utilization was increased to 50%. As in Figures 8.1 and 8.2, the *DVMA* family of algorithms performed better than the two benchmarks. The **CCDVMA** algorithm did have a higher average delay than the other two *DVMA*-based algorithms, but this can be attributed to selection of its initial tree based on tree cost and not delays.

Overall, the *DVMA* family of algorithms performed 10-20% better in average delay than **KPP** and **BSMA**. This means that these algorithms are suitable for solving very tight delay bound, Δ, constraints.

8.2 Delay Variation

Because **KPP** and **BSMA** were not designed to satisfy the *delay variation* constraint, it is not surprising that the *DVMA* family of algorithms performed better

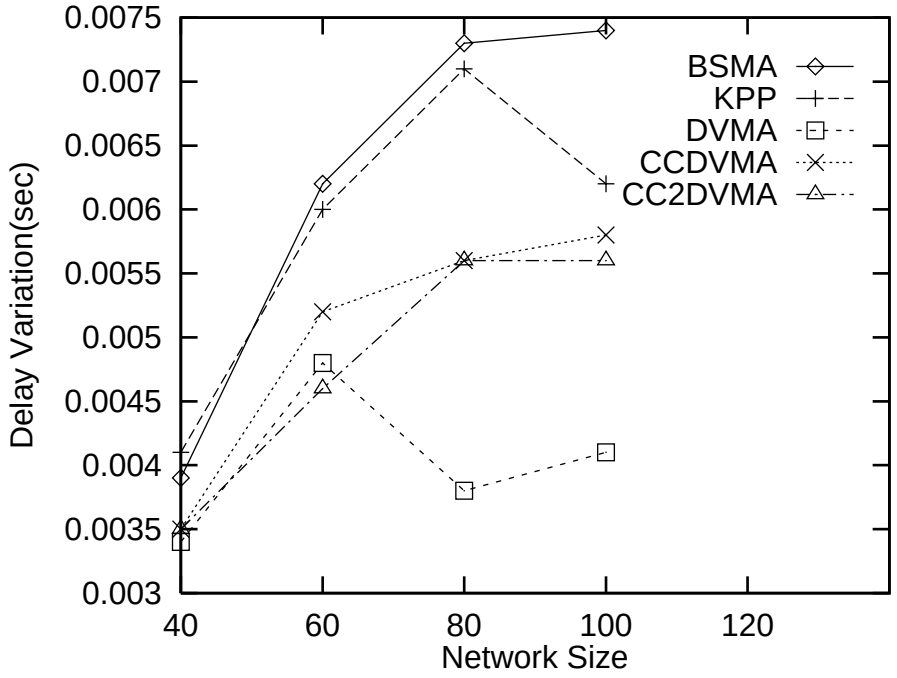

Figure 8.4: Delay Variation Comparison for 5% Multicast Group membership and 15% network utilization

for this constraint. As expected, **DVMA** performed the best for this measure,see Figure 8.4. **CCDVMA** and **CC2DVMA** offer a better average case performance than **KPP** and **BSMA**.

In Figure 8.5 and Figure 8.6, it appears that **CCDVMA** degrades in performance for the delay variation constraint. However, the tree generation failure rates presented in Section 8.4 will show that **KPP** and **BSMA** are showing statistically skewed results due to the fact that they are generating fewer trees than the other algorithms. This is important because these average results are only obtained from runs that successfully generated *feasible* multicast trees.

On average, the *DVMA* family of algorithms performed well in meeting the delay variation constraints. In isolated cases, **KPP** and **BSMA** were able to meet the constraint, but overall performed much worse than the other algorithms.

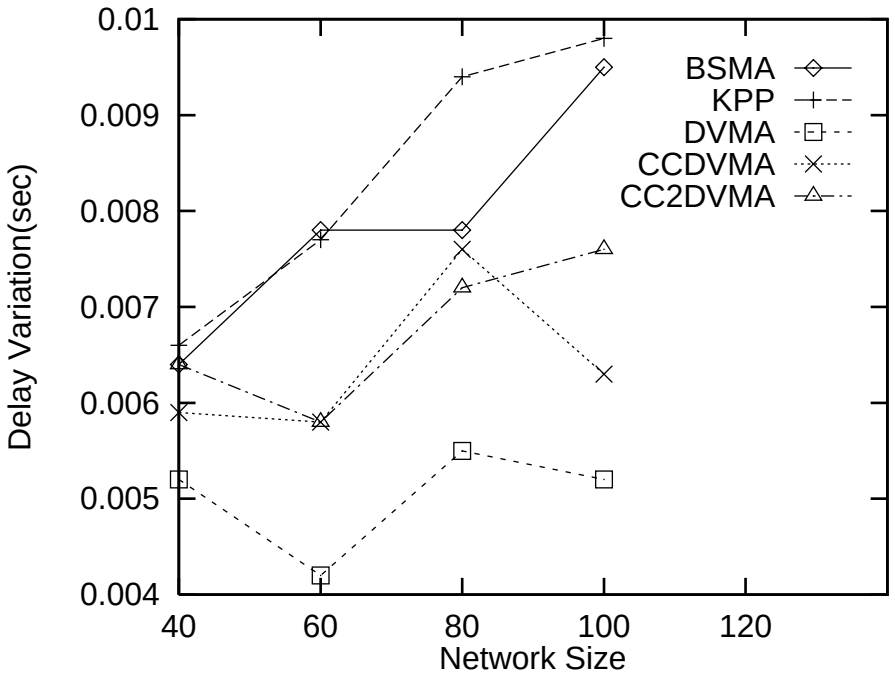

Figure 8.5: Delay Variation Comparison for 10% Multicast Group membership and 15% network utilization

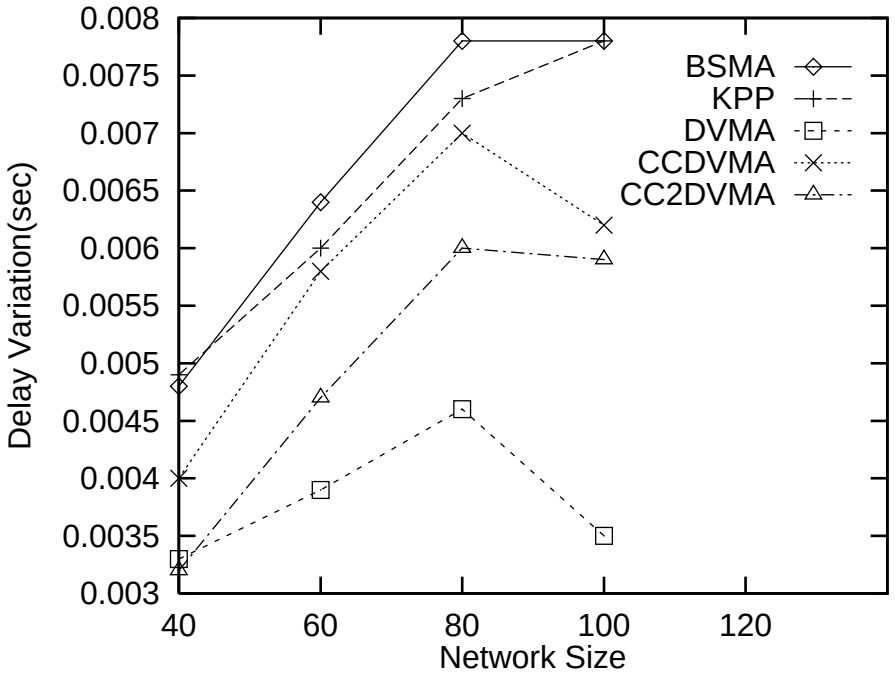

Figure 8.6: Delay Variation Comparison for 5% Multicast Group membership and 50% network utilization

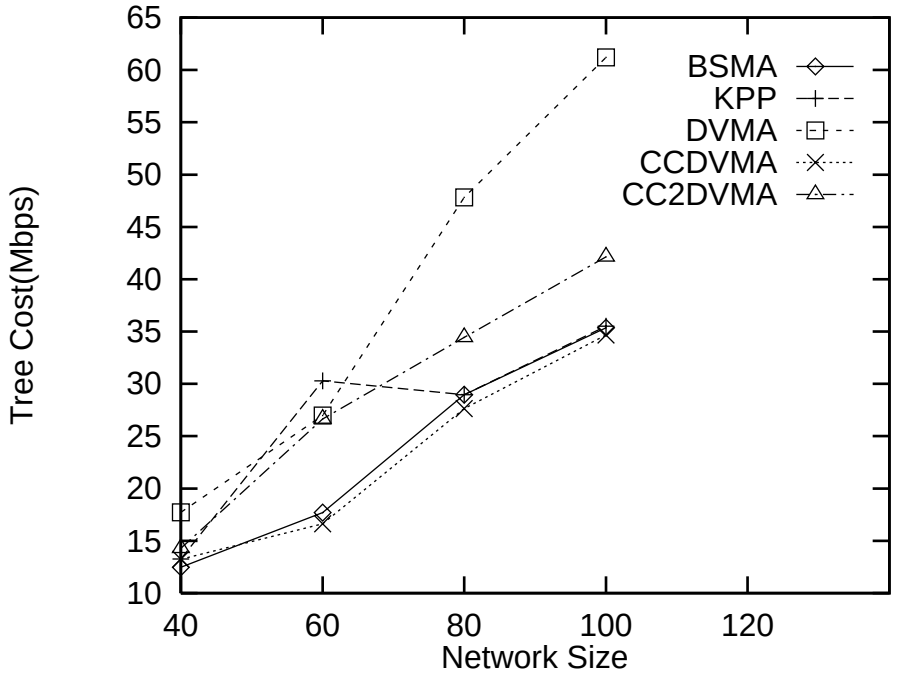

Figure 8.7: Cost Comparison for 5% Multicast Group membership and 15% network utilization

8.3 Cost

The cost comparison of the five algorithms is the primary basis for this research. The data presented here shows very favorable results for **CCDVMA** and **CC2DVMA**.

In Figure 8.7, **CCDVMA** and **CC2DVMA** generate multicast trees that have costs that are very competitive with **KPP** and **BSMA**. In fact, for this set of experimental variables, **CCDVMA** actually has better cost than any of the other algorithms. Remember that **CCDVMA** starts with an initial tree that is the tree of lowest cost generated by Dijkstra's shortest path algorithm. When the **CCDVMA** search algorithm runs, it only replaces tree T with T_i if it has lower cost and simply meets the delay and delay variation constraints. This is similar to the approach used by **BSMA**. The results here show that **CCDVMA** is performing as well as **BSMA** in reducing the overall tree cost while still meeting the delay constraint.

As in Figure 8.7, Figures 8.8 and 8.9 show that **CCDVMA** and **CC2DVMA**

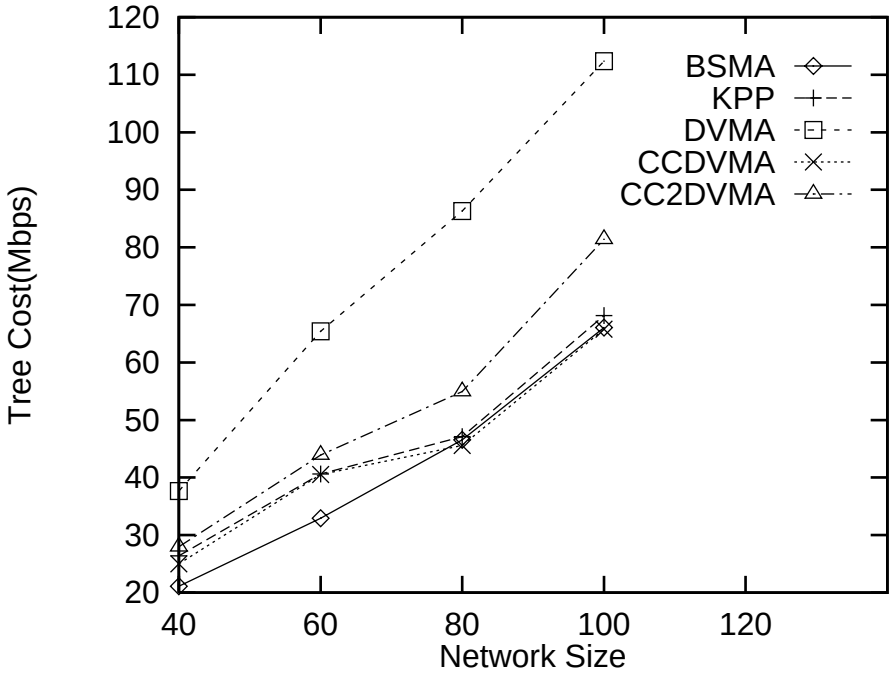

Figure 8.8: Cost Comparison for 10% Multicast Group membership and 15% network utilization

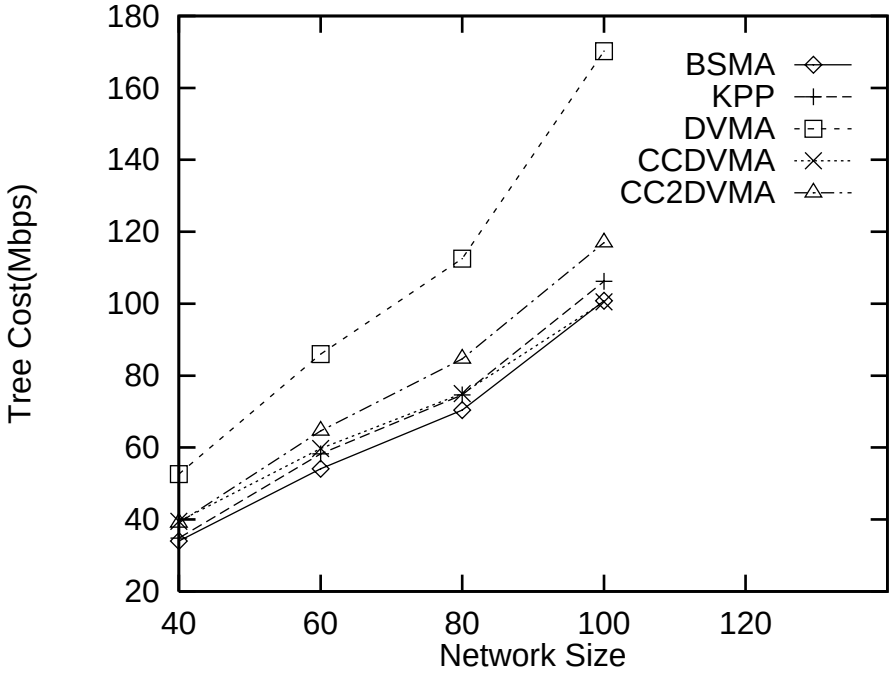

Figure 8.9: Cost Comparison for 5% Multicast Group membership and 50% network utilization

perform very well in minimizing the cost of the multicast trees. **CC2DVMA**, on average, has a 5-10% higher cost than **CCDVMA**, **KPP**, and **BSMA**. Again, **CCDVMA** is generating trees of similar cost as the two benchmark algorithms.

These cost results show that **CCDVMA** is a competitive solution in minimizing the the cost of multicast trees. It should be remembered that these cost numbers for **CCDVMA** include more sample points than **KPP** or **BSMA**. These means that the average cost of **CCDVMA** is a better statistical average than the average cost of the benchmarks.

8.4 Multicast Tree Generation Failure Rate

The measure of tree generation failure rate is important for two reasons. The first is that if an algorithm cannot generate a *feasible* multicast tree for a high majority of user constraints, the user is left in a difficult situation. Either the constraints must be re-negotiated, which may take a long period of time, or the multicast session cannot be completed and the user cannot complete the task. Neither situation is favorable. The second reason for this measurement is to *combine* the results from the previous sections.

A multicast session can fail to be completed for two reasons. Either the delay constraint Δ could not be met, or the delay variation constraint δ could not be satisfied. To the user, regardless of the reason, the multicast session could not complete. The user is generally only concerned with whether or not the session could be established. For that reason, we incorporated the failure rate measure in our results.

Overall, **DVMA** had the lowest failure rate of the five algorithms. In Figures 8.10 and 8.12, **CCDVMA** had comparable failure rate values. The other three algorithms generally had much higher failure rates. A majority of these are due to violations of Equation 4.2. The results depicted in Figure 8.11 show that under some situations, all five algorithms can have a high failure rate. This can be caused by several reasons. The user constraints could be very tight. The network topology also has an affect on the algorithms ability to meet user constraints. In general, since the CCDVBMT problem is $\mathcal{NP}$-complete, for any given network, it may be infeasible to test all

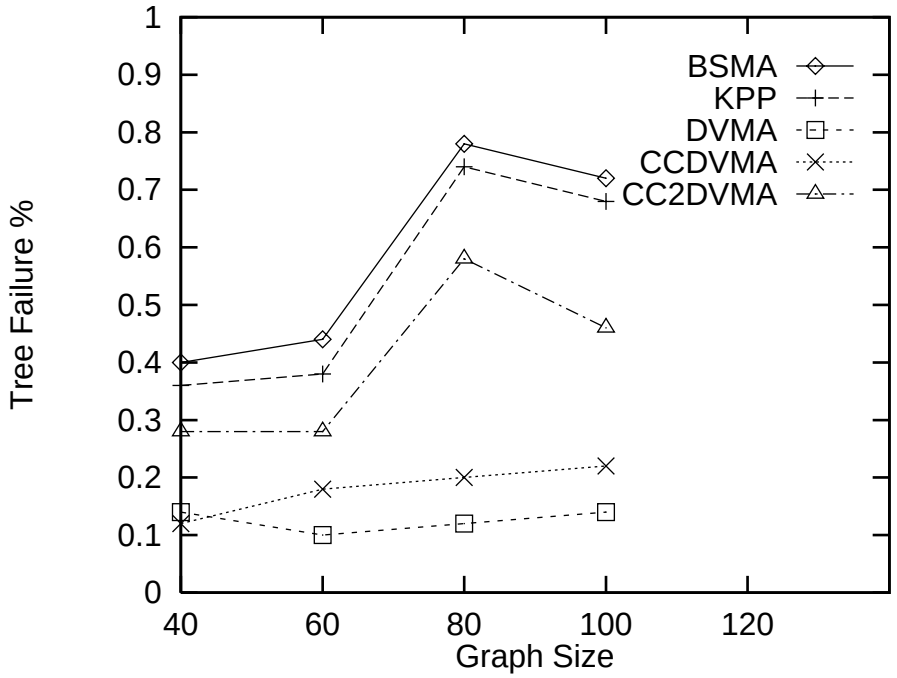

Figure 8.10: Tree Generation Failure Percentage for 5% Multicast Group membership and 15% network utilization

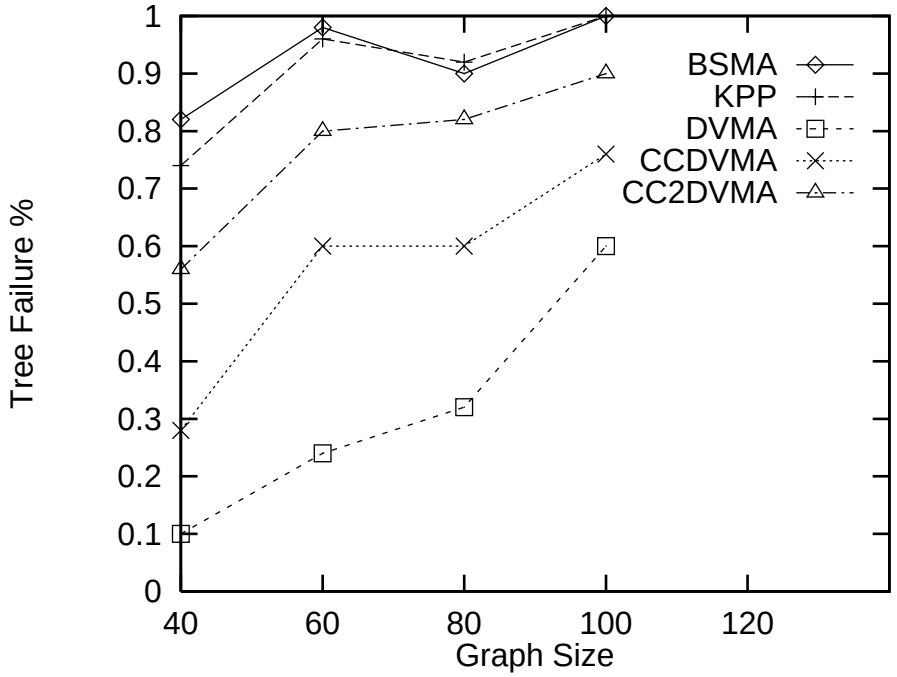

Figure 8.11: Tree Generation Failure Percentage for 10% Multicast Group membership and 15% network utilization

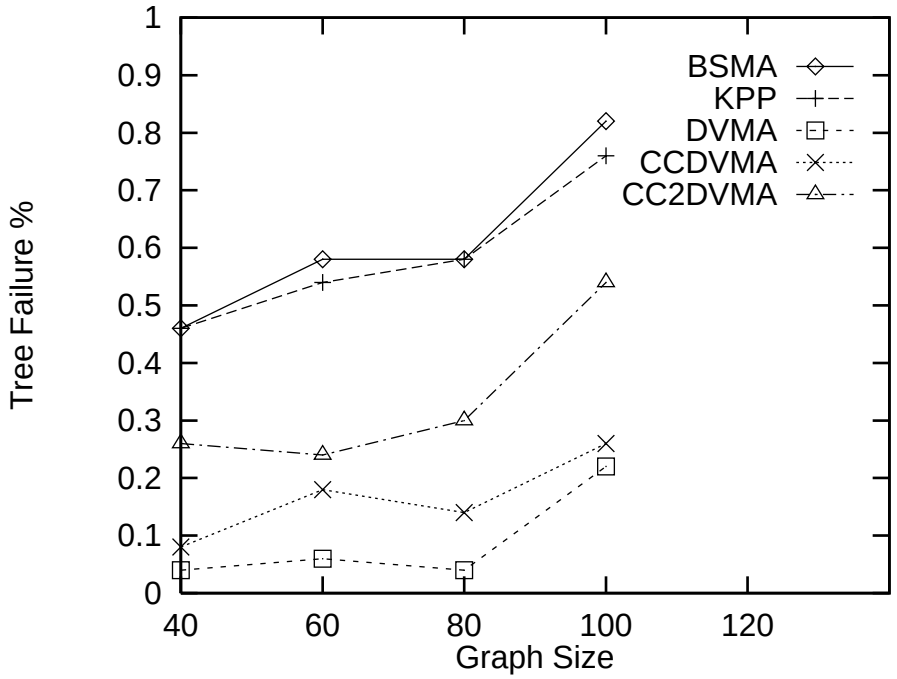

Figure 8.12: Tree Generation Failure Percentage for 5% Multicast Group membership and 50% network utilization

possible multicast trees.

8.5 System Running Time

Ramanathan presented an algorithm that contains a *control knob* that allows the algorithm to be *tuned* for tradeoffs between running time and tree cost [14]. Because running time can be a factor in the usability of an algorithm, it was included in this study as a comparative measure. We excluded the system running time numbers for **KPP** and **BSMA** from this section. An explanation for this can be found in Appendix C.

The numbers presented in these figures represent the average system time each algorithm used to calculate a multicast tree for a single network. The system time only includes the time spent in the actual algorithm computing the multicast tree.

Figure 8.13 shows the running time of the *DVMA* family of algorithms. It is not surprising that **CC2DVMA** has a higher running time than the other two. As

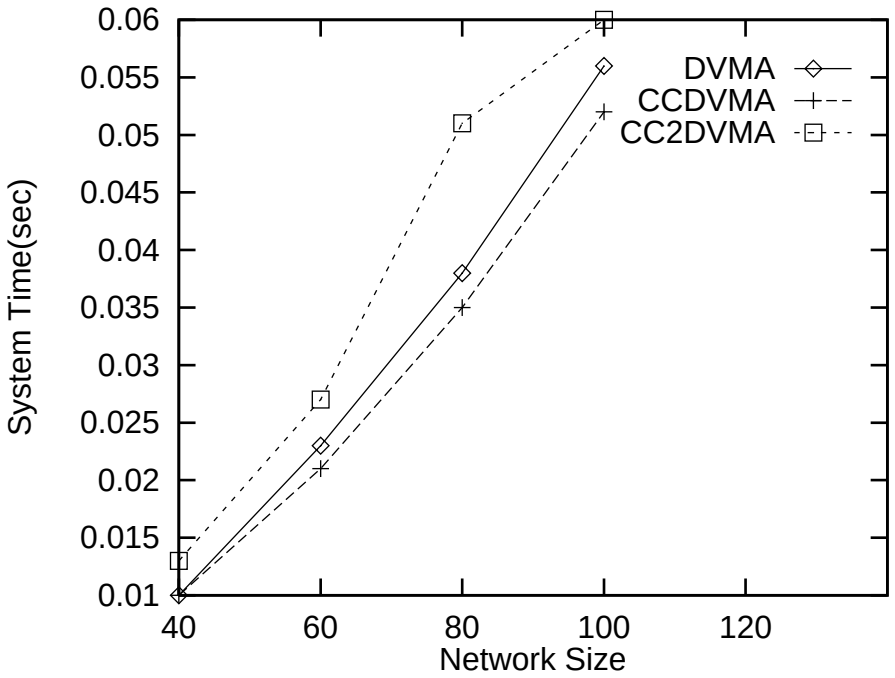

Figure 8.13: Average Running Time for 5% Multicast Group membership and 15% network utilization

Algorithm	40 Nodes	60 Nodes	80 Nodes	100 Nodes
DVMA	4.9	3.7	2.8	2.0
CCDVMA	4.2	3.5	2.7	1.9
CC2DVMA	5.3	4.1	3.0	2.3

Table 8.1: Average Number of Multicast Trees Generated per network for 5% Multicast Group and 15% Traffic Utilization

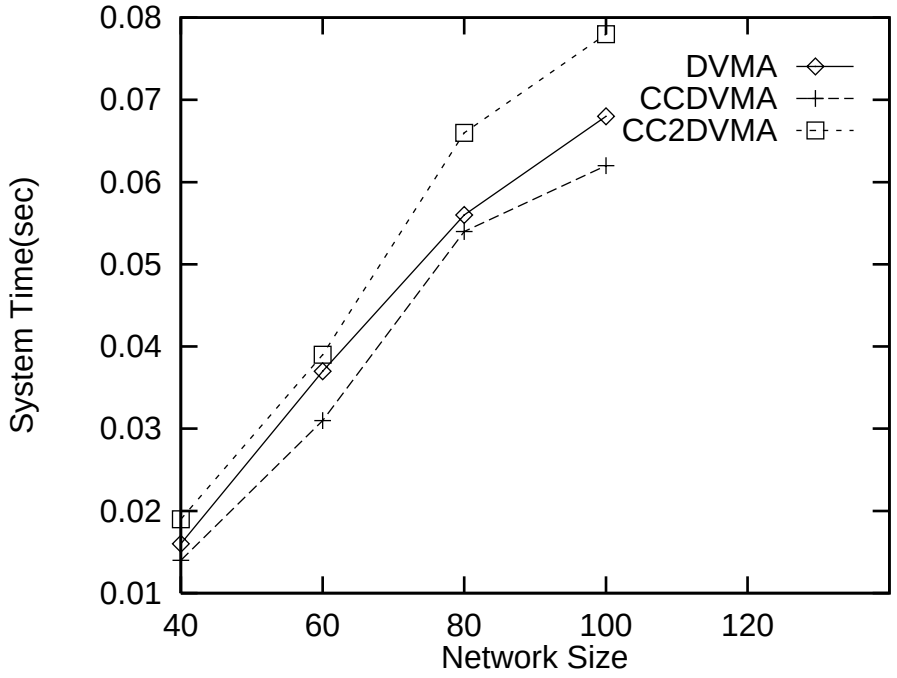

Figure 8.14: Average Running Time for 10% Multicast Group membership and 15% network utilization

explained in Section 5.4, **CC2DVMA**'s average case complexity is the same as its worst-case complexity. It must test every tree that it generates in order to find the tree of least cost. The fact that **CCDVMA** performs better than **DVMA** can be explained by the logic used to choose the k-shortest paths on line 2 of Figure 5.1. In general, **CCDVMA** chooses less paths than **DVMA** during this step. This in turn causes less trees to be tested. Similar results can be seen in Figure 8.14.

In order to make a more thorough comparison, it was necessary to determine the

Algorithm	40 Nodes	60 Nodes	80 Nodes	100 Nodes
DVMA	4.5	3.3	2.2	1.6
CCDVMA	4.2	3.0	1.9	1.6
CC2DVMA	4.7	3.5	2.4	1.9

Table 8.2: Average Number of Multicast Trees Generated per network for 10% Multicast Group and 15% Traffic Utilization

reason for the differences in the system running time. The three algorithms compared in this section all have the same general code flow. The primary cause of differences in system running time is the number of potential multicast trees that each algorithm generates. Table 8.1 shows the average number of multicast trees that are generated per simulated network. The results shown in Table 8.1 correspond to the system running times graphed in Figure 8.13. It can be seen that there is a direct correlation between these two measures. A similar comparison can be made between Figure 8.14 and Table 8.2.

8.6 Major Results

The performance of **CCDVMA** is encouraging. The results of this research show that **CCDVMA** is adept at balancing the user constraints with the demands of the underlying network. As the use of real-time interactive applications increases, this ability will become increasingly more important.

Overall, **CCDVMA** did not consistently outperform the other algorithms in any given measurement, though its system running time is encouraging. It did perform well in all measures. These results show that **CCDVMA** is a fast heuristic for solving the intractable CCDVBMT problem. It also scales well to larger network topologies and still provides very good routes. None of the other algorithms used in this research gave better overall results.

Appendix B presents additional results from this work. These results were compiled to verify findings by Rouskas and Baldine [15] concerning the average node degree and to test varying values of the delay variation constraint δ.

Chapter 9

Summary and Future Work

9.1 Summary

We have considered the problem of generating cost conscious multicast trees that satisfy the delay and delay variation constraints imposed by the user process. The **CCDVMA** heuristic that we developed exhibits good average case behavior across a variety of network topologies. Our simulation environment was setup to model typical multicast scenarios over high-speed networks. This allowed us to make valid comparisons between our new heuristics and a representative set of existing algorithms.

The results presented in Section 8 are promising. The **CCDMVA** heuristic offers a favorable solution to the CCDVBMT problem. The average cost of its multicast trees is comparable to that of the more established **KPP** and **BSMA** algorithms. The average path delay results are better than those posted by **BSMA** and **KPP** and compare favorably with **DVMA**. It offers a good balance between tree cost, average delay, and average delay variation, while being very competitive in terms of system running time.

We have also addressed the *dynamic* reorganization of multicast trees in response to changes in the multicast group. The time complexity of these solutions is competitive with those of **DVMA**.

The **CCDVMA** heuristic is a novel strategy for addressing the constrained Steiner tree problem. In our work, this approach has shown promise, not only in offering a

solution to the CCDVBMT problem, but doing so while exhibiting good performance numbers.

9.2 Future Work

There are still several possibilities for the extension of our work. The *delay variation* constraint is still a new concept. The ability to minimize the inter-destination delay is becoming more important as new distributed applications are developed. Within the *DVMA* family of algorithms, there are still uninvestigated means of minimizing the tree cost. Other constrained Steiner tree algorithms could also be modified to utilize the delay variation constraint.

This algorithm requires that each node have complete knowledge of the network. In addition, routers in the network have no knowledge of the multicast communication. In general, routers will have up to date information about changes in the network that will affect the multicast sessions. For these reasons, a next logical step would be to develop a distributed version of this algorithm that runs on the routers. Routers would be able to add/delete low cost paths as nodes dynamically join/leave the multicast session. This would relieve the nodes from having to collect topology information as well as actually knowing who is participating in the multicast group.

Some of the results of this work not presented in Section 8, but included in Appendix C, show that the system running time of the benchmark algorithms are rather high. Additional investigation is needed to determine if this phenomenon was introduced in the simulation environment or is inherent to the algorithms.

Our results suggest that it is possible that **CCDVMA** could be an efficient solution to the constrained Steiner tree problem. Additional research needs to be carried out to explore this possibility. The preliminary results are encouraging, but not conclusive.

Bibliography

[1] S. Baase. *Computer Algorithms.* Addison Wesley, 1988.

[2] I. Baldine. Multicast routing with end-to-end delay and delay variation constraints. Master's thesis, North Carolina State University, Department of Computer Science, 1995.

[3] D. Bertsekas and R. Gallager. *Data Networks.* Prentice Hall, Inc., 1992.

[4] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[5] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Co., New York, NY, 1979.

[6] S. L. Hakimi. Steiner's problem in graphs and its implications. *Networks*, pages 1:113–133, 1971.

[7] A. Koifman and S. Zabele. RAMP: A reliable adaptive multicast protocol. *Proceedings of IEEE Infocom '96*, pages 1442–1451, March 1996.

[8] V. P. Kompella, J. C. Pasquale, and G. C. Polyzos. Multicast routing for multimedia communication. *IEEE/ACM Transactions on Networking*, 1(3):286–292, June 1993.

[9] L. Kou, G. Markowsky, and L. Berman. A fast algorithm for steiner trees. *Acta Informatica*, pages 15:141–145, 1981.

[10] E. Lawler. *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehart and Winston, 1976.

[11] J. Lin and S. Paul. RMTP: A reliable multicast transport protocol. *Proceedings of IEEE Infocom '96*, pages 1414–1424, March 1996.

[12] M. R. Macedonia and D. P. Brutzman. Mbone provides audio and video across the internet. *IEEE Computer*, 27:30–36, April 1994.

[13] S. Paul, K. K. Sabnani, and D. M. Kristol. Multicast transport protocols for high speed networks. *Proceedings of International Conference on Network Protocols*, pages 4–14, 1994.

[14] S. Ramanathan. An algorithm for multicast tree generation in networks with asymmetric links. *Proceedings of IEEE Infocom '96*, pages 337–344, March 1996.

[15] G. Rouskas and I. Baldine. Multicast routing with end-to-end delay and delay variation constraints. *IEEE Journal on Selected Areas in Communications*, 15(3), April 1997.

[16] H. Salama. *Multicast Routing for Real-time Communication on High-Speed Networks*. PhD thesis, North Carolina State University, Department of Electrical and Computer Engineering, Dec 1996.

[17] H. Salama, D. Reeves, Y. Viniotis, and T. Sheu. Comparison of multicast routing algorithms for high-speed networks. Technical Report IBM-TR29.1930, IBM, September 1994.

[18] H. Salama, D. Reeves, Y. Viniotis, and T. Sheu. Evaluation of multicast routing algorithms for distributed real-time applications on high-speed networks. *Proceedings of 6th IFIP Conference on High Speed Networks*, pages 27–42, September 1995.

[19] H. Salama, Y. Viniotis, D. Reeves, and T. Sheu. Multicast routing algorithms for high-speed networks. Technical Report IBM-TR29.1883, IBM, May 1994.

[20] J. S. Turner. New directions in communications (or which way to the information age?). *IEEE Communications Magazine*, 24(10):8–15, Oct 1986.

[21] B. Waxman. Routing of multipoint connections. *Journal on Selected Areas in Communications*, 6(9):1617–1622, December 1990.

[22] R. Widyono. The design and evaluation of routing algorithms for real-time channels. Technical Report TR-94-024, University of California at Berkeley, Department of EECS, June 1994.

[23] Q. Zhu, M. Parsa, and J. J. Garcia-Luna-Aceves. A source-based algorithm for near-optimum delay-constrained multicasting. *Proceedings of IEEE Infocom '95*, pages 377–385, March 1995.

Appendix A

Experimental Measurements for Variables k and l

Rouskas and Baldine stated that the natural upper bound on the parameters k and l is the maximum number of paths of delay at most Δ [15]. In the development of **DVMA**, artificial limits were imposed on these parameters. The upper bounds are : $k = 3n$ and $l = n$. However, they expected the maximum value of these parameters to be small constants. The data presented in this section verifies this assumption.

During the simulation phase, the maximum value and the average value of k and l was measured. Originally, this was done to verify the findings dealing with the system running time of the algorithms under test. However, it also proved that, in general, k and l do not increase the run-time complexity of the *DVMA* family of algorithms.

Table A.1 shows the maximum and average value of k over the simulation run for 5% multicast group membership and 15% traffic utilization. The data shows that **DVMA** and **CCDVMA** generate only a small number of potentially *feasible* trees. **CC2DVMA** generates more trees due to its search logic.

In Table A.2, the average value of l shows that the algorithms are only able to generate a small number of paths at line 10 in Figure 5.1. The complexity of the algorithm on line 10 was shown to be $\mathcal{O}(ln^3)$, so the parameter l adds very little to the overall complexity. For **CC2DVMA**, the maximum value of l approaches the upper limit of n, but on average it is still much lower than the limit.

Algorithm	40 Nodes	60 Nodes	80 Nodes	100 Nodes
DVMA	*Max* 7.0	*Max* 7.0	*Max* 6.0	*Max* 9.0
	Avg 4.9	*Avg* 3.7	*Avg* 2.8	*Avg* 2.0
CCDVMA	*Max* 5.0	*Max* 4.0	*Max* 4.0	*Max* 2.0
	Avg 4.2	*Avg* 3.5	*Avg* 2.7	*Avg* 1.9
CC2DVMA	*Max* 16.0	*Max* 10.0	*Max* 9.0	*Max* 20.0
	Avg 5.3	*Avg* 4.1	*Avg* 3.0	*Avg* 2.3

Table A.1: Upper limit and average values of k for 5% Multicast Group and 15% Traffic Utilization

Algorithm	40 Nodes	60 Nodes	80 Nodes	100 Nodes
DVMA	*Max* 23.0	*Max* 32.0	*Max* 23.0	*Max* 28.0
	Avg 5.1	*Avg* 13.3	*Avg* 10.1	*Avg* 14.5
CCDVMA	*Max* 5.0	*Max* 11.0	*Max* 14.0	*Max* 20.0
	Avg 3.2	*Avg* 10.3	*Avg* 9.8	*Avg* 13.6
CC2DVMA	*Max* 32.0	*Max* 51.0	*Max* 73.0	*Max* 88.0
	Avg 20.4	*Avg* 30.9	*Avg* 26.3	*Avg* 28.9

Table A.2: Upper limit and average values of l for 5% Multicast Group and 15% Traffic Utilization

Algorithm	40 Nodes	60 Nodes	80 Nodes	100 Nodes
DVMA	*Max* 5.0	*Max* 7.0	*Max* 7.0	*Max* 11.0
	Avg 4.5	*Avg* 3.3	*Avg* 2.2	*Avg* 1.6
CCDVMA	*Max* 5.0	*Max* 4.0	*Max* 3.0	*Max* 3.0
	Avg 4.2	*Avg* 3.0	*Avg* 1.9	*Avg* 1.6
CC2DVMA	*Max* 18.0	*Max* 22.0	*Max* 10.0	*Max* 17.0
	Avg 4.7	*Avg* 3.5	*Avg* 2.4	*Avg* 1.9

Table A.3: Upper limit and average values of k for 10% Multicast Group and 15% Traffic Utilization

Algorithm	40 Nodes	60 Nodes	80 Nodes	100 Nodes
DVMA	*Max* 17.0	*Max* 35.0	*Max* 38.0	*Max* 29.0
	Avg 4.3	*Avg* 11.0	*Avg* 12.2	*Avg* 9.9
CCDVMA	*Max* 6.0	*Max* 13.0	*Max* 17.0	*Max* 19.0
	Avg 3.0	*Avg* 8.8	*Avg* 10.2	*Avg* 7.8
CC2DVMA	*Max* 38.0	*Max* 55.0	*Max* 68.0	*Max* 74.0
	Avg 18.8	*Avg* 30.0	*Avg* 27.7	*Avg* 24.8

Table A.4: Upper limit and average values of l for 10% Multicast Group and 15% Traffic Utilization

In Tables A.3 and A.4, the average values of k and l are actually lower than those values in Tables A.1 and A.2. We attribute this to the increase of the multicast group size causing increased path sharing. The increase in path sharing adds more destinations to the tree per pass of the search algorithm.

Overall, these tables show that the assumption that k and l are small constants is a good assumption. In addition, the time complexity of the *DVMA*-based algorithms does not increase because of these parameters. Another observation that can be made from this data is that, on average, there is a small number of possible paths from the source to any destination given the delay constraints imposed on the multicast tree. This means that search algorithms like **CCDVMA** and **DVMA** may not have to impose artificial upper limits on these parameters.

Appendix B

Additional Simulation Results

The following figures show additional research findings. The findings presented here are included to re-enforce the findings of Section 8. Section B.1 shows the results of lowering the average node degree from 4 to 2.5. Section B.2 provides the results of raising the delay variation constraint from 0.01 seconds to 0.02 seconds.

B.1 Results for Node Degree 2.5

It was found that the *DVMA* family of algorithms performed better as the average node degree increased [15, 2]. We attempt to verify these findings by running a set of simulations with the average node degree reduced to 2.5.

It should be noted that the trend today in high-speed networks is towards a high average node degree. These results do show that as the node degree decreases, the performance of all the algorithms converges. This is to be expected. As the node degree decreases, the number of possible paths decreases, thus limiting the path choices that can be made.

The following figures verify that the performance of the *DVMA*-based algorithms deteriorate as the node degree decreases. However, it should be noted that **KPP** and **BSMA** fail to generate *any* multicast trees for any of the 100 node networks.

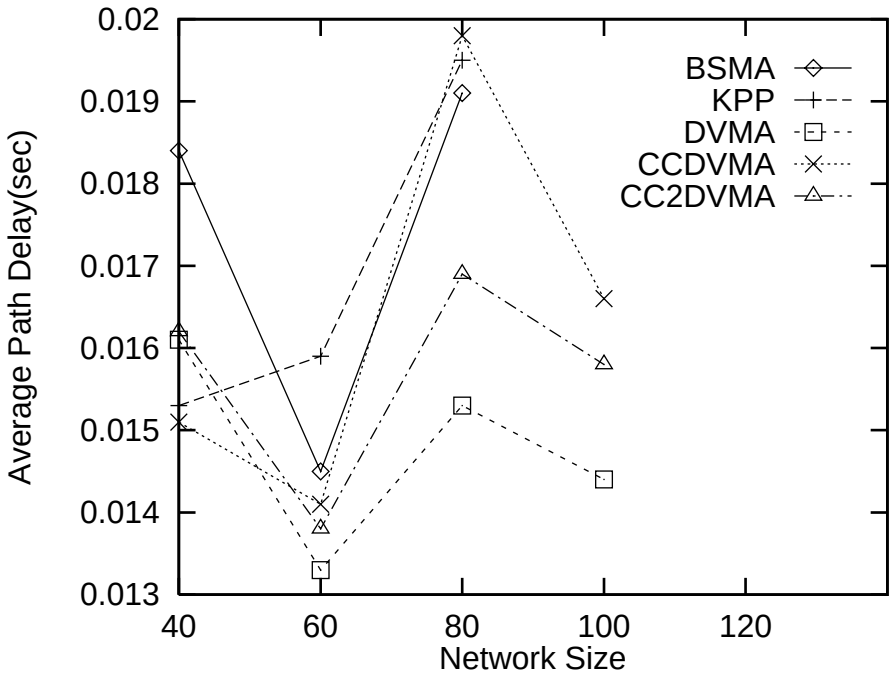

Figure B.1: Average Path Delay for 5% Multicast Group membership, 15% network utilization, and Average Node Degree = 2.5

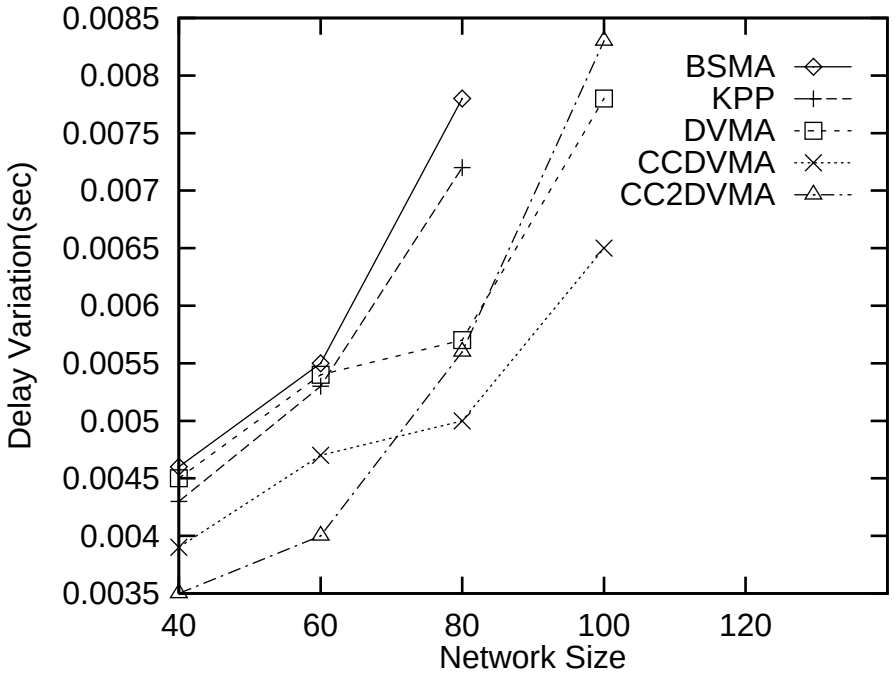

Figure B.2: Delay Variation Comparison for 5% Multicast Group membership, 15% network utilization, and Average Node Degree = 2.5

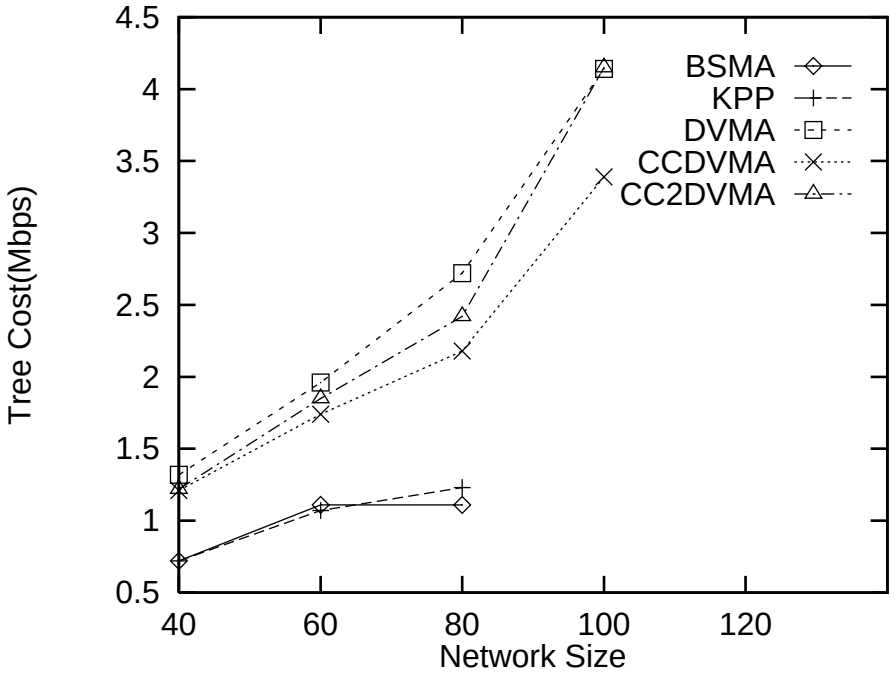

Figure B.3: Cost Comparison for 5% Multicast Group membership, 15% network utilization, and Average Node Degree = 2.5

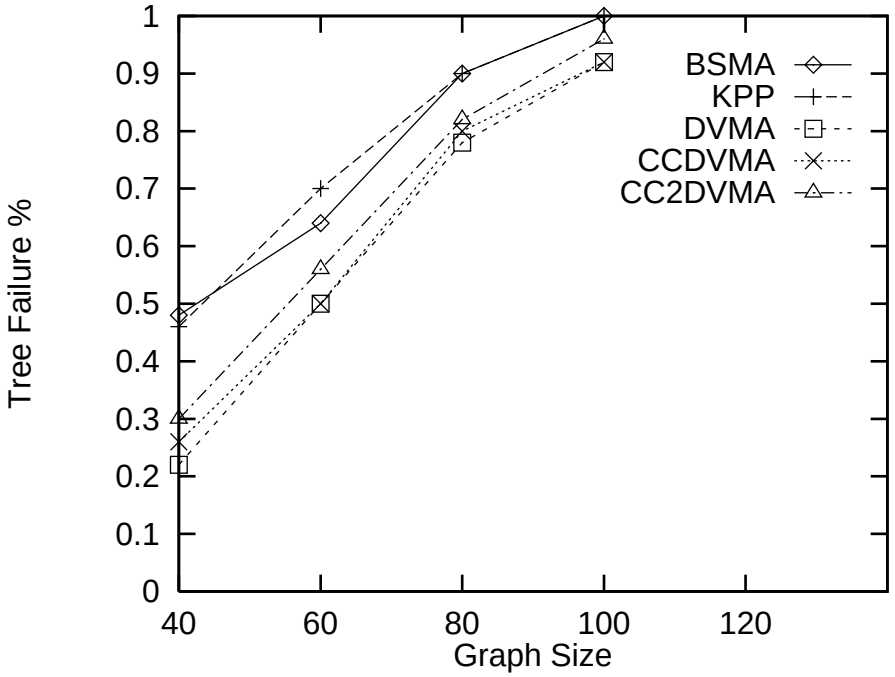

Figure B.4: Tree Generation Failure % for 5% Multicast Group membership, 15% network utilization, and Average Node Degree = 2.5

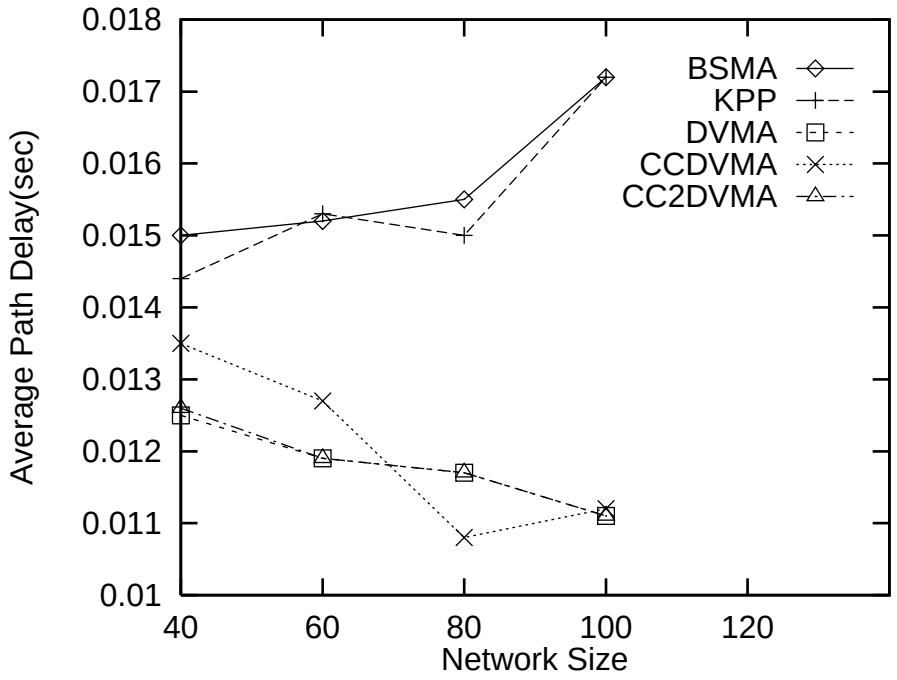

Figure B.5: Average Path Delay for 10% Multicast Group membership, 50% network utilization, and Delay Variation = 0.02 seconds

B.2 Results for Delay Variation Constraint 0.02

When a *feasible* tree is unobtainable with the current user constraints, the first course of action would be to re-negotiate the constraints. The following figures show the results of increasing the delay variation constraint from 0.01 seconds to 0.02 seconds.

In Figure B.6, it can be seen that as the delay variation constraint is loosened, the average delay variation for all five algorithms begin to converge. This result is not that surprising. The benchmark algorithms, **KPP** and **BSMA**, will benefit from the looser constraint and have a better opportunity to meet the delay variation target. With the *DVMA*-base algorithms, the looser constraint makes it more likely that the initial tree T_0 will satisfy (4.1) and (4.2). The selection of T_0 will return to the user the tree of least cost or least delay and this tree will be more similar to the tree returned by **KPP** or **BSMA**.

It should be noted that in Figure B.7 the average tree cost for **DVMA** is much

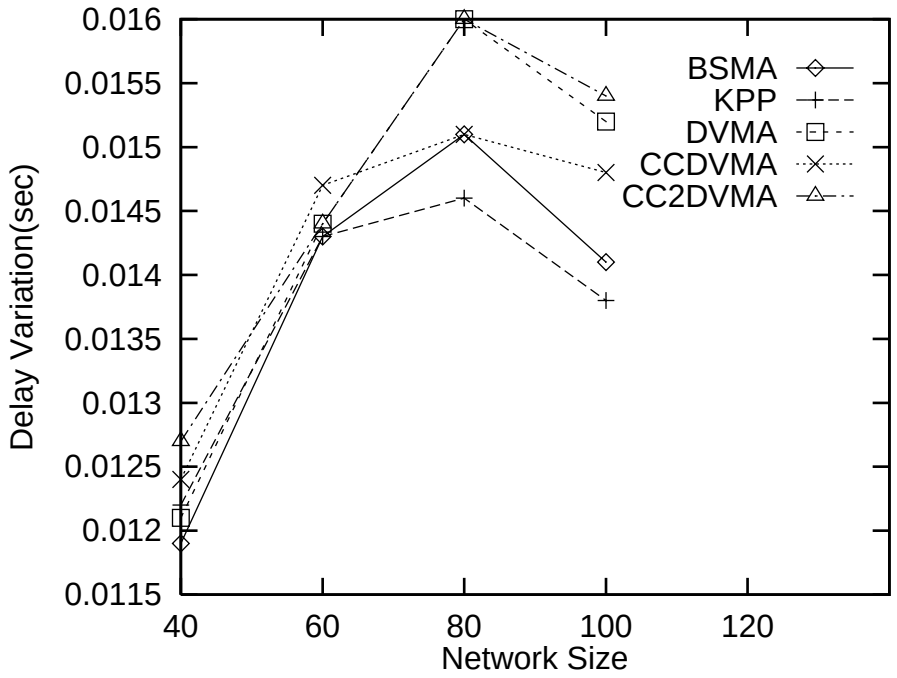

Figure B.6: Delay Variation Comparison for 10% Multicast Group membership, 50% network utilization, and Delay Variation = 0.02 seconds

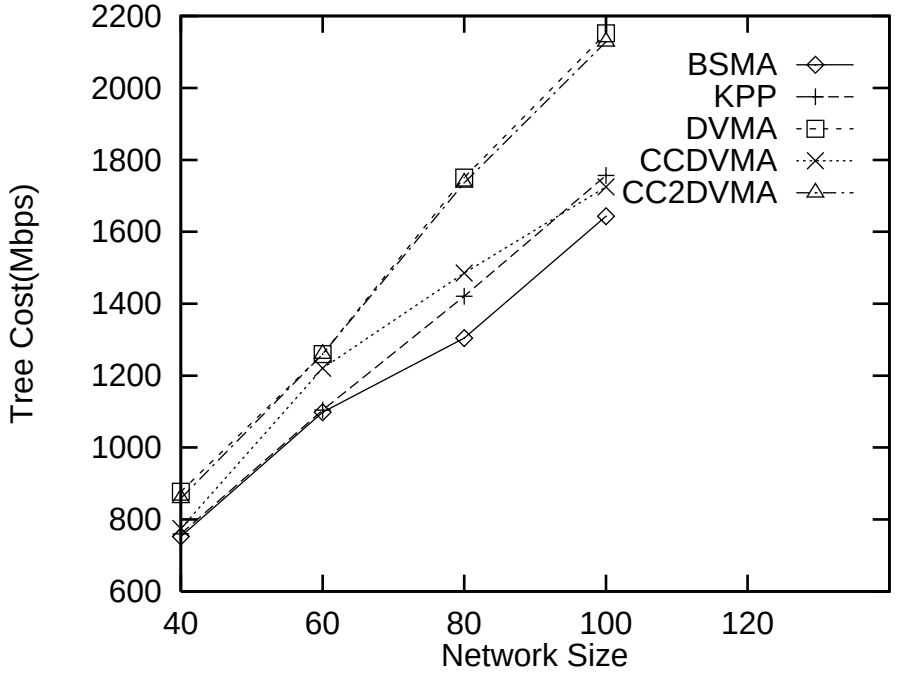

Figure B.7: Cost Comparison for 10% Multicast Group membership, 50% network utilization, and Delay Variation = 0.02 seconds

closer to the costs of the benchmark algorithms. This could warrant further investigation since **DVMA** never attempts to minimize the tree cost. The path selection mechanism could be selecting good delay paths which have correspondingly low utilization components.

Appendix C

System Running Time Results for BSMA **and** KPP

In the process of measuring the performance of the five algorithms, we found that **KPP** and **BSMA** were using an unusually high amount of CPU time. Figures C.1 and C.2 show that the system running time of **KPP** and **BSMA** are significantly higher than the other three.

One possible cause could be the implementation differences of the algorithms. The **MCRSIM** package is implemented in C++. The *DVMA* algorithms were implemented mostly in C and integrated into the simulator package. It is not clear how much of the system time measurement is due to the added overhead of C++. One piece of follow-on research will try to eliminate the variable of C versus C++ implementation and get a more accurate comparison of the running time of the five algorithm.

It is also possible that **BSMA** could be exhibiting this performance due to the number of possible replacement *superedges* it has to choose from. The work done by Salama [16] obtained system running time numbers on graphs with a small number of nodes (≤ 20). Our work takes these measures over much larger graphs. The increase in the number of nodes introduces more possible paths from a source to a destination. **DVMA** and **CCDVMA** limit the number of paths they can choose from to $3n$, but the results in Appendix A show that this maximum number is never a factor. **BSMA**

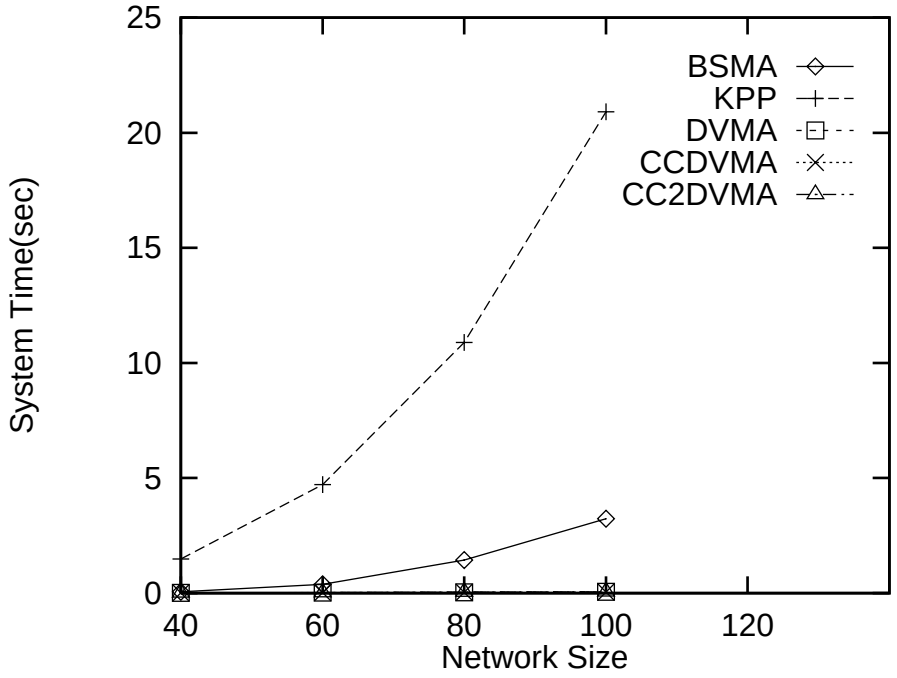

Figure C.1: System Running Time for 5% Multicast Group membership and 15% network utilization

could be generating more replacement edges than **DVMA** and **CCDVMA** as the network size increases.

As far as **KPP** is concerned, the problem may be related to the delay constraint Δ. When **KPP** computes its closure graph, the fact that Δ is not an integer could have an impact on the running time. In the description of **KPP** [8], it is stated that an assumption is made that Δ is integer. If the assumption is false, the computation of the constrained all-pairs could take a significantly longer amount of time.

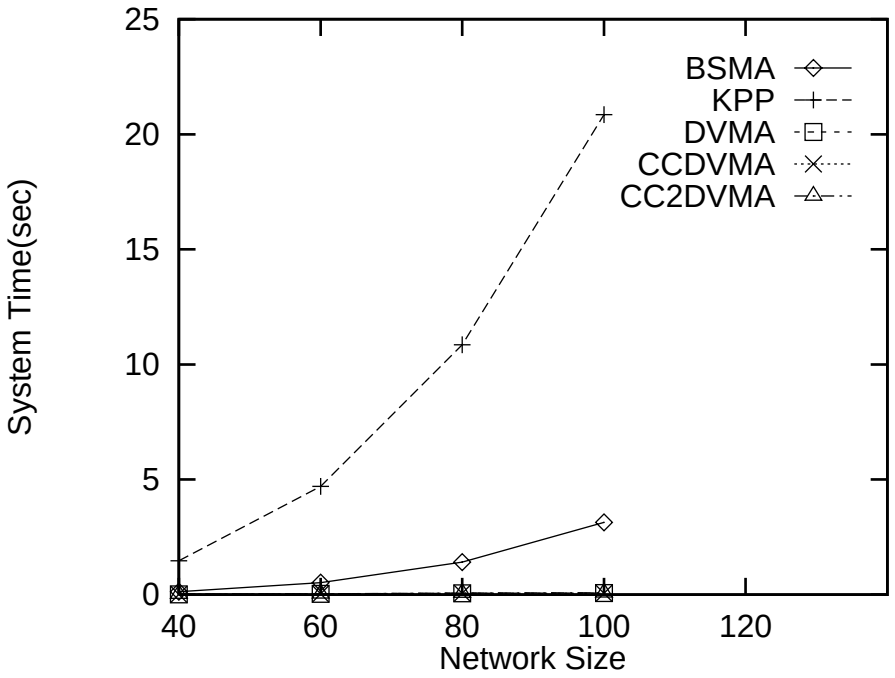

Figure C.2: System Running Time for 10% Multicast Group membership and 15% network utilization

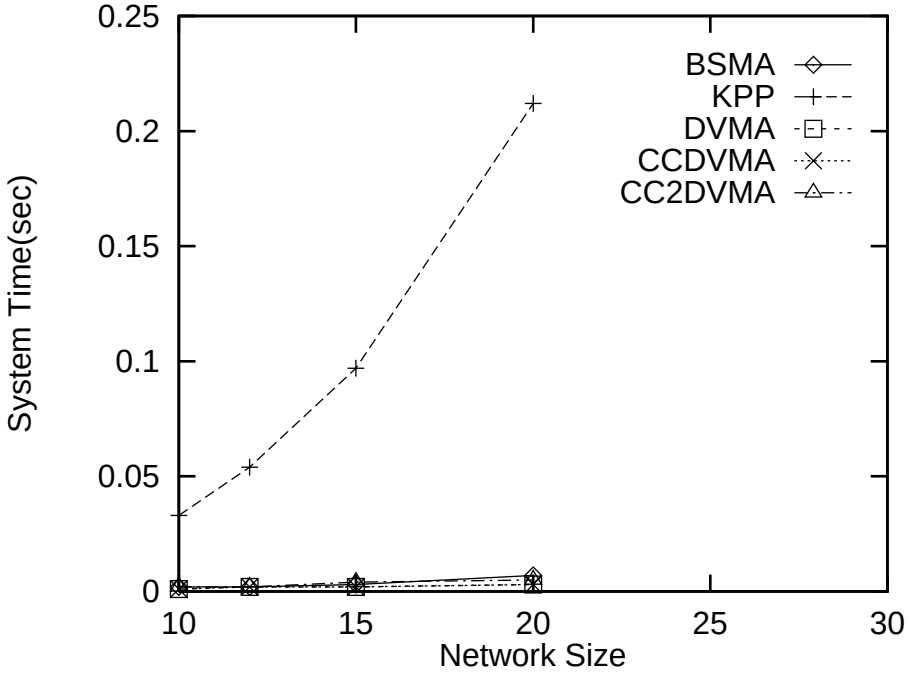

Figure C.3: System Running Time for 5% Multicast Group membership and 15% network utilization on smaller networks