Université d'Ottawa • University of Ottawa

PERMISSION DE REPRODUIRE ET DE DISTRIBUER LA THÈSE

PERMISSION TO REPRODUCE AND DISTRIBUTE THE THESIS

NOM DE L'AUTEUR / NAME OF AUTHOR:	Keqin ZHU
ADRESSE POSTALE / MAILING ADDRESS:	49 Woodpark Way Nepean, ON K2J 4C6
GRADE / DEGREE:	**ANNÉE D'OBTENTION / YEAR GRANTED**
Ph.D.(Computer Science)	2003
TITRE DE LA THÈSE / TITLE OF THESIS:	
QoS-based multicast routing for real-time communications	

L'auteur permet, par la présente, la consultation et le prêt de cette thèse en conformité avec les règlements établis par le bibliothécaire en chef de l'Université d'Ottawa. L'auteur autorise aussi l'Université d'Ottawa, ses successeurs et cessionnaires, à reproduire cet exemplaire par photographie ou photocopie pour fins de prêt ou de vente au prix coûtant aux bibliothèques ou aux chercheurs qui en feront la demande.

Les droits de publication par tout autre moyen et pour vente au public demeureront la propriété de l'auteur de la thèse sous réserve des règlements de l'Université d'Ottawa en matière de publication de thèses.

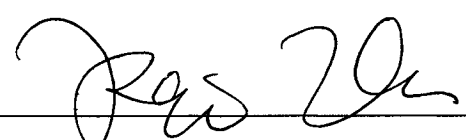
The author hereby permits the consultation and the lending of this thesis pursuant to the regulations established by the Chief Librarian of the University of Ottawa. The author also authorizes the University of Ottawa, its successors and assignees, to make reproductions of this copy by photographic means or by photocopying and to lend or sell such reproductions at cost to libraries and to scholars requesting them.

The right to publish the thesis by other means and to sell it to the public is reserved to the author, subject to the regulations of the University of Ottawa governing the publication of theses.

N.B. LE MASCULIN COMPREND ÉGALEMENT LE FÉMININ

Mar. 21, 2003

DATE



(AUTEUR) SIGNATURE (AUTHOR)

Université d'Ottawa · University of Ottawa

Université d'Ottawa · University of Ottawa

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTDOCTORALES

FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES

ZHU, Keqin

AUTEUR DE LA THÈSE - AUTHOR OF THESIS

Ph.D. (Computer Science)

GRADE - DEGREE

School of Information Technology and Engineering

FACULTÉ, ÉCOLE, DÉPARTEMENT - FACULTY, SCHOOL, DEPARTMENT

TITRE DE LA THÈSE - TITLE OF THE THESIS

QoS-Based Multicast Routing for Real-Time Communications

Hasan Ural

DIRECTEUR DE LA THÈSE - THESIS SUPERVISOR

EXAMINATEURS DE LA THÈSE - THESIS EXAMINERS

P. Flocchini

O. Yang

E. Kranakis

I. Matta

J.-M. De Koninck, Ph.D.

LE DOYEN DE LA FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTDOCTORALES

SIGNATURE

DEAN OF THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES

QoS-based multicast routing for real-time communications

by

Keqin Zhu

A thesis submitted to the Faculty of Graduate and Post-Doctoral Studies in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Computer Science

Ottawa-Carleton Institute for Computer Science
School of Information Technology and Engineering
University of Ottawa
Ottawa, Ontario, Canada

Abstract

This thesis studies the distributed multicast routing algorithms that provide quality of service (QoS) guarantees in networks. It focuses on the delay constrained minimum cost tree (or constrained Steiner tree) problem; that is, it constructs a multicast tree that not only meets the constrained end-to-end delay requirements but also is the minimum cost multicast tree. This problem is known to be NP-complete.

The existing distributed multicast routing algorithms that are based on heuristics do not consider a network environment where node failures occur. As a result, these algorithms will fail to complete the construction of a multicast tree when node failures occur during the construction period. Moreover, they will fail to maintain the constructed multicast tree if node failures occur after the construction period and during the on-going multicast session. In both cases, these algorithms will have to be restarted. We propose two new distributed delay constrained multicast routing algorithms that provide QoS guarantees even in networks where node failures occur. One is shortest path (SP) based and the other is minimum spanning tree (MST) based. They are capable of constructing a delay constrained multicast tree when node failures occur during the tree construction period and recovering from any node failure in a multicast tree during the on-going multicast session without interrupting the running traffic on the unaffected portion of the tree. The proposed SP-based or MST-based algorithms perform the failure recovery efficiently, which give better performance in terms of the number of exchanged messages and the convergence time than the existing distributed SP-based or MST-based delay constrained multicast routing algorithms in a network where node failures occur, respectively.

The existing distributed delay constrained multicast routing algorithms can be classified either as an MST-based heuristic or as a SP-based heuristic. Two representative algorithms i.e., DKPP and DSPH are MST and SP based heuristics, respectively. The MST based algorithms like DKPP run with a high message and time complexity of $O(n^3)$. Furthermore, the MST based algorithms have a very low success rate in constructing a constrained multicast tree especially under tight delay constraints. On the other hand, the SP based algorithms like DSPH have a fatal deficiency; that is, these algorithms will not be able to find a solution if the delay constraint is less than the maximum delay of a cost based shortest path tree. Both types of algorithms try to construct the multicast tree sequentially without taking advantage of the concurrency in a distributed algorithm. We propose an efficient distributed delay constrained multicast routing algorithm which builds the multicast tree in a concurrent manner rather than in a sequential manner. As a result, the proposed algorithm runs with a very low message and time complexity and has a near zero failure rate in constructing a multicast tree even under tight delay constraints without any limitation on the value of delay constraints.

The proposed "concurrent" algorithm is then augmented with a fault recovery mechanism that efficiently constructs a multicast tree in networks where node failures occur. Finally, the proposed "concurrent" algorithm is expanded into a distributed delay constrained dynamic multicast routing algorithm that handles the dynamic multicast membership effectively and efficiently.

Acknowledgments

First, I would like to thank my supervisor, Dr. Hasan Ural, for his support and guidance through the course of this work. Especially, I would like to express my sincere appreciation of his great accessibility as Dr. Ural always quickly responded to my requests for help by offering his advice and feedback.

I would also like to thank the members of my Ph.D. committee, Dr. Evangelos Kranakis of Carleton University and Dr. Paola Flocchini of the University of Ottawa for their suggestions on how to improve this work.

I would also like to thank the members of the Telecommunications Software Research Group and the Distributed Computing Research Group during this research. Especially, Dr. Robert L. Probert and Dr. Gregor v. Bochmann taught the courses that sparked my interest in this research, Dr. Ivan Stojmenovic provided suggestions on the selection of research topics, Dr. Alan Williams and graduate student Igor Sales and David Whittier attended my presentation and provided feedback on this work. I would also like to thank Louise Desrochers for providing administrative assistance.

I would also like to thank my managers, Carol Chamberlain and Kathleen Ryan at Nortel Networks for their support.

I would also like to thank Prof. X. Jia at City University of Hong Kong, Prof. D.S. Reeves at North Carolina State University, and Prof. B. M. Waxman at Southern Illinois University for providing their simulation software to us.

I especially would like to thank my family for their support and encouragement: my wife Ling Zhang, my son Sandford and my daughter Angela. Without their sacrifice, I would not be able to complete this work.

Last, but not least, I gratefully acknowledge the support of: the Natural Sciences and Engineering Research Council of Canada (NSERC), Ontario Ministry of Education and Training and the University of Ottawa.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

Multicast is the simultaneous transmission of data from a source to a group of destinations. There are many multimedia applications (e.g., video conferencing and distance education) which require multicast communication. Moreover, multimedia applications are often high bandwidth and delay sensitive, which require real-time communication with low end-to-end delays [SRV97][Pau98].

The routing function is defined as consisting of two parts [BG92]. The first part selects a route for the session during the connection establishment phase, and the second part ensures that each packet of that session is forwarded along the assigned route. In this thesis, only the route selection algorithms are considered. During the connection establishment phase, the system selects a route along which sufficient resources ought to be reserved to meet the quality of service (QoS) requirements specified by applications, such as network bandwidth and maximum message delay. This is an important step to guarantee real-time data to be delivered to destinations with an acceptable delay.

Multicast routing is to find a routing tree (called *multicast tree*) which is rooted from a source node s and contains all nodes in a set S of destination nodes. There are two reasons for basing efficient multicast routes on trees: i) the data can be transmitted concurrently to various destinations along the branches of the tree; and ii) a minimum

number of copies of the data are transmitted, with replication of data being necessary only at forks in the tree.

There are two important requirements on algorithms for constructing multicast trees. The first is the constrained end-to-end delay along the individual paths from source to each destination where the delay constraint is specified by the application performing the multicast. The second is the minimum cost of the multicast tree. The cost of the multicast tree is the sum of the costs on the edges in the multicast tree. For example, the costs on the edges in the multicast tree can be in terms of network bandwidth utilization. The minimum cost tree is called a *Steiner tree*, and the problem of finding a Steiner tree is known to be NP-complete [Kar72]. The delay-constrained minimum cost tree is called a *constrained Steiner tree* [KPP93]. The problem of finding a constrained Steiner tree is also NP-complete [KPP92].

Algorithms based on heuristics for constructing constrained Steiner trees have been developed in recent years. Some algorithms are classified as *source-based* (or *centralized*) multicast routing algorithms if the source is assumed to have all the information necessary to construct the multicast tree, such as KPP [KPP93], CDKS [Sun95], CKMB [Sun98], CAO [Wid94], and BSMA [ZPG95]; while others are classified as *distributed* multicast routing algorithms [BV96], if it is not required to maintain the entire network status information in each node, and multiple nodes are participating in constructing the multicast tree, such as DKPP [KPP96][KPP93a] and DSPH [Jia98]. Distributed routing algorithms are usually slow and complex. On the other hand, source-based routing algorithms are not practical for very large networks, since they assume complete knowledge of the network on the source node.

In addition to the algorithms mentioned above, there are also many QoS-based protocols that have been proposed in the literature, such as YAM (or spanning join) [CC97], QoSMIC [FBP98] and QMRP [CNS00]. They are receiver(destination)-initiated and based on the multiple-path approach; that is, the protocols will generate multiple paths between the existing multicast tree and a receiver, and then the receiver will pick the best one based on certain QoS criteria to connect itself into the tree. In the following, we will focus on discussing the algorithms for constructing constrained Steiner trees, which are source-initiated and do not use the multiple-path approach.

The distributed multicast routing algorithms are initiated from one node; that is the source node. The application running on the source node requires that a constrained multicast tree rooted at the source node to be constructed so that data (e.g., multimedia data) can be sent to all destination nodes.

The distributed multicast routing algorithms can be further classified as minimum spanning tree (MST) or shortest path (SP) based heuristic algorithms. For example, DKPP is an MST based algorithm, while DSPH is an SP based algorithm. In an MST based multicast routing algorithm, a distributed MST algorithm as described in [GHS83] which mimics Prim's MST algorithm [CLR92][Prim57] is used to construct a multicast tree with additional checking to ensure that the delay constraint is not violated during the expansion of the tree. The algorithm will stop when all destination nodes are reached and those branches that do not reach any destination nodes will be pruned. In an SP based multicast routing algorithm, the multicast tree is expanded by a delay constrained shortest path from the tree to one destination node at a time until all destination nodes are covered. A *delay*

constrained shortest path is the path with the minimum cost among all paths whose delays are under the delay constraint.

We found that there are three main problems associated with the distributed multicast routing algorithms. The first problem is that they do not take into account the changes in the topology of the network [BV96]. Whenever the topology of the network changes, these algorithms will fail to complete the construction of a multicast tree and it is up to the application to re-start the algorithm. As a result, these algorithms will have a low success rate for the construction of a multicast tree (which is the number of trials to build a multicast tree successfully divided by the number of total trials). Similarly, as these algorithms do not respond to the changes in the topology of the network after the multicast tree is built, it is up to the application to restart the algorithm to re-build the tree for the changed network topology, which causes the interruption of the running traffic for all existing members in a multicast session.

The second problem with the distributed multicast routing algorithms is that either MST-based or SP-based algorithms try to construct the multicast tree sequentially without taking advantage of the concurrency in a distributed algorithm. Thus, they take a long time to construct a multicast tree with low efficiency.

The third problem with the distributed multicast routing algorithms is that the known solution to the dynamic multicast routing problem (i.e., dynamic membership) requires high number of message exchanges for including a new multicast group member into the multicast tree.

In this thesis, we propose several new distributed delay-constrained multicast routing algorithms to address the topology change problem with efficient fault recovery

mechanisms, to solve the low-efficiency problem with a concurrent solution, and to handle the dynamic membership problem with an effective and efficient dynamic multicast routing algorithm.

1.2 Network model and problem definition

A network is modeled as a connected, directed graph $G=(V, E)$ where nodes in node set V represent network nodes and edges in edge set E represent links. In addition, there are two values associated with each link e ($e \in E$): delay $D(e)$ and cost $C(e)$. It is assumed that both values do not change during a multicast session. The link delay $D(e)$ is the delay a packet experiences on the corresponding link, the link cost $C(e)$ is a measure of the utilization of the corresponding link's resources. Links are asymmetrical, namely the cost and delay for the link $e = (i, j)$ and the link $e' = (j, i)$ may not be the same.

The constrained Steiner tree (CST) problem (or delay-constrained multicast tree problem) can be stated as follows. Given a network $G = (V, E)$ with n nodes, a source node s ($s \in V$), a set of m destination nodes S ($S \subseteq V - \{s\}$) called *multicast group*, and a delay constraint Δ, find a tree T ($T \subseteq G$) rooted at s that spans the nodes in S such that

i) for each node v in S, the delay on the path from s to v is bounded above by Δ; that is, $\sum_{e \in P(s, v)} D(e) < \Delta$ where $P(s, v)$ is the unique path in T from s to v, and

ii) the tree cost $\sum_{e \in T} C(e)$ is minimized.

This tree is called a *constrained Steiner tree* (or *delay constrained multicast tree*).

Some definitions that will be used later are given below. *A shortest path from a tree T to a non-tree node v* is a shortest path (in terms of cost) from a tree node u to v, denoted by $SP(u, v)$, and u satisfies

$\sum_{e \in SP(u, v)} C(e) \leq \sum_{e \in SP(k, v)} C(e)$ for any tree node k.

The cost of the shortest path from a tree T to a non-tree node v is called *the cost from a tree T to a non-tree node v*, and denoted by $C(T, v)$. A non-tree node v is said to be *the closest to a tree T* if it satisfies $C(T, v) \leq C(T, w)$ for any non-tree node w.

1.3 Related work

1.3.1 Steiner tree and multicast routing algorithm

The multicast routing problem can be modeled as a Steiner tree problem, which tries to find a multicast tree with the minimum tree cost without considering any other constraints. The Steiner tree (ST) problem (or multicast routing problem) can be formulated as follows. Given a network $G=(V, E)$ with n nodes, a source node s ($s \in V$), and a set of m destination nodes S ($S \subseteq V - s$) (each of which is called *group member*), find a tree T ($T \subseteq G$) rooted at s that spans the nodes in S such that the tree cost

$\sum_{e \in T} C(e)$ is minimized.

The resulted tree is called a *Steiner tree* (or *multicast tree*). The Steiner tree problem was shown to be NP-complete [Kar72]. So, all proposed efficient algorithms for solving the Steiner tree problem are heuristics.

Waxman is one of the pioneers who applied and evaluated various heuristics for the Steiner tree problem to the multicast routing problem [Wax88][Wax93]. The most well-known Steiner tree heuristics are KMB heuristic [KMB81] and TM heuristic [TM80]. Given graph $G = (V, E)$ with n nodes, a source node s ($s \in V$), a set of m destination nodes S, KMB heuristic starts with constructing a complete graph $G' = (V, E')$ where each edge represents a shortest path between the two nodes in G and its cost is the length of the shortest path. Then it constructs a minimum spanning tree (MST) T' for G'.

Next it converts the edges in T' to paths in G forming G_S which may not be a tree since the expanded paths may cross each other, and finds an MST T_S of G_S. Finally it prunes tree branches which do not lead to any nodes in $S \cup \{s\}$ to obtain a solution T. On the other hand, TM heuristic starts with putting a single source node s in the tree. Then, it finds a destination node which can be reached by the shortest path (SP) from the existing tree T', and adds the destination node along with the SP into the tree at each step until all destination nodes in S are included in the tree. Both KMB heuristic and TM heuristic are centralized algorithms. They employ a different combination of MST and SP algorithms to achieve a good solution for the Steiner tree problem. Both heuristics construct a tree with a cost that is within twice that of the Steiner tree. As KMB heuristic relies more on an MST algorithm while TM heuristic relies more on an SP algorithm, we can classify KMB heuristic and TM heuristic as MST-based and SP-based heuristic respectively.

Table 1.1 Multicast routing algorithms (Steiner tree algorithms)

Algorithm	**Features**	**Time complexity**	**Tree cost**	**References**
KMB	• MST-based	$O(mn^2)$	$\leq 2(1-1/l)$ times that of the optimal tree, where l is the number of leaves in the optimal tree	[KMB81]
TM	• SP-based • greedy algorithm based on Prim's MST	$O(mn^2)$	$\leq 2(1-1/m)$ times that of the optimal tree	[TM80]
RS	• Kruskal's MST-based	$O(n^3)$	$\leq 2(1-1/m)$ times that of the optimal tree	[RS83][RC86] [WI88]
BJ	• MST-based	$O(mn^2)$	$\leq 2(1-2/(m+1))$ times that of the optimal tree	[BJ83]

In addition to the two heuristics discussed above, another important Steiner tree heuristic algorithm is RS heuristic [RS83], which is based on Kruskal's MST algorithm [Kru56]. It starts with a set of single node sub-trees consisting of nodes in $S \cup \{s\}$, finds a

node $v \in V$ with the best average distance to all current sub-trees, and then merges two sub-trees that are closest to v by a shortest path through v. It repeats the merging operation until a single tree is obtained. With higher time complexity, RS heuristic tends to give better performance in terms of tree cost on average [RC86]. More heuristics are discussed in surveys on Steiner tree [Win87][HR92] and the comparison of various heuristics' computational performance are conducted in experimental evaluations on heuristic methods [WS92][Vos92]. Table 1.1 summarizes these Steiner tree algorithms which have been frequently referred to and applied in solving the multicast routing problem.

Many Steiner tree heuristics are directly derived from the research of the multicast routing problem. Bharath-Kumar and Jaffe's algorithm [BJ83] is similar to KMB heuristic but without applying the MST algorithm again on G_S. Maxemchuk [Max97] proposed a modified TM heuristic to solve the multicast routing problem for video distribution, where the link cost is associated with the maximum rate of the receivers that share the same link. By separating the receivers into subsets according to their required rate and covering them in the order from the highest rate subset to the lowest rate subset, the modified TM heuristic has better performance in terms of tree cost than the original TM heuristic.

Centralized Steiner tree (or multicast routing) algorithms are not practical for very large networks, since they assume complete knowledge of the network on the source node. Bauer and Varma [BV96] proposed two distributed algorithms for the Steiner tree problem, both of which are SP-based algorithms. But they differ in the way that the tree is built. One starts with a forest of multicast group members, and then connects them in pair into successively larger sub-trees until a single multicast tree is obtained. It is basically the

distributed version of a modified RS heuristic. The other starts with the source node in the tree and grows the tree by connecting with one group member a time until all multicast group members are included in the tree. This is the distributed version of the TM heuristic.

1.3.2 Constrained Steiner tree and QoS-based multicast routing algorithm

The solution for the Steiner tree problem only takes tree cost into consideration. As a result, the trees generated as the solution for the Steiner tree problem may violate other constraints such as the end-to-end delay along the individual paths from source to each destination.

The QoS-based multicast routing problem is to find a multicast tree with the minimum tree cost under the required constraints (i.e., QoS requirements). One of the important QoS requirements is the end-to-end delay from the source to each destination. The delay-constrained minimum cost tree is called *constrained Steiner tree* (or *constrained multicast tree*). For the sake of clarity, the original multicast routing problem is also called *unconstrained* multicast routing problem and its generated tree is called *unconstrained* multicast tree. The constrained Steiner tree problem has also been shown to be NP-complete [KPP92].

Many algorithms based on heuristics for constructing constrained Steiner trees have been proposed in literature in recent years. Most of them are classified as *source-based* (or *centralized*) multicast routing algorithms such as KPP [KPP93], CDKS [Sun95], CKMB [Sun98], CAO [Wid94], BSMA [ZPG95], and DCMA [ZKC01], since it is assumed that the source has all the information necessary to construct the multicast tree. Some of them are classified as *distributed* multicast routing algorithms, in which it is not required to maintain the entire network status information in each node, and multiple

nodes are participating in constructing the multicast tree, such as DKPP [KPP96][KPP93a] and DSPH [Jia98].

KPP algorithm [KPP93] is similar to KMB heuristic but takes the delay constraint into consideration. It starts with constructing a complete graph $G' = (V, E')$ where each edge represents a delay-constrained shortest path between the two nodes in G and its cost is the length of the shortest path. Then it constructs a constrained spanning tree (MST) T' for G', which minimizes the tree cost as much as possible under the delay constraint. Finally it converts the edges in T' to paths in G and prunes tree branches which do not lead to any node in $S \cup \{s\}$ to obtain a solution T.

CDKS algorithm [Sun95] gives a method for constructing delay-constrained multicast trees based on the Dijkstra's shortest path algorithm [Dij59] by first trying to construct a constrained low cost tree and then computing the delay-based shortest path tree for those multicast group nodes which are not in the constructed constrained multicast tree yet. This algorithm is called *Constrained Dijkstra's (CDKS)* as it is based on the Dijkstra's shortest path algorithm.

Like KPP algorithm, CKMB algorithm [Sun98] is based on KMB heuristic [KMB81], a heuristic for Steiner tree problem, by adding constraints into KMB heuristic's first two steps. So, this algorithm is called *Constrained KMB (CKMB)*. The main goal of CKMB algorithm is to reduce the $O(n^3\Delta)$ time complexity of KPP algorithm by applying a different algorithm in KPP algorithm's first 2 steps. In the first step, instead of using a dynamic programming approach similar to Floyd's shortest path algorithm [Flo62] to calculate the constrained shortest path, CKMB heuristic uses a greedy approach similar to Dijkstra's shortest path algorithm to calculate a *constrained path*, of which the delay is

bounded above by a certain constraint. It is the same algorithm as that used in CDKS for constructing a constrained low cost tree. In the second step, instead of using a greedy approach similar to Prim's minimum spanning tree algorithm to construct a constrained spanning tree T, CKMB uses the same greedy algorithm used in the first step except the selection criterion is different. As a result, CKMB has a time complexity of $O(mn^2)$, which is better than KPP's $O(n^3\Delta)$. Meanwhile, the study shows that the cost of trees generated by CKMB is about the same as that by KPP.

The constrained adaptive ordering (CAO) algorithm [Wid94] is based on the constrained Bellman-Ford shortest path algorithm which takes delay-constraints into consideration. It uses the constrained Bellman-Ford shortest path algorithm to find a destination node which has the minimum cost constrained path from the existing sub-tree to the destination node, and to connect the destination node into the existing sub-tree until all destination nodes are included in the tree. To be able to find the minimum cost constrained path from the existing sub-tree to a destination node, the costs of links in the already existing sub-tree are set to zero before applying the constrained Bellman-Ford shortest path algorithm each time. The strategy of this algorithm is similar to the TM heuristic with the added delay constraint checking.

The Bounded Shortest Multicast Algorithm (BSMA) [ZPG95][ZPG98] starts by computing a delay-based shortest path tree (SPT) T for the given source s and multicast group S. It defines a *relay node* as a node with degree equal to 2 and not a node in $\{s\} \cup S$, and a *super-edge* as the longest simple path in T between 2 non-relay nodes. Then it iteratively replaces a super-edge in T with a lower cost super-edge without violating the

delay bound, until the cost of the tree can no longer be reduced. A *k*-shortest path algorithm [Law76, pp. 100-104] is used to find a lower cost super-edge.

According to the performance evaluation conducted in [SRV97], BSMA gives the best performance in terms of the tree cost. Table 1.2 summarizes the centralized delay-constrained multicast routing algorithms.

Table 1.2 Centralized delay-constrained multicast routing algorithms

Algorithm	**Features**	**Time complexity**
KPP [KPP92] [KPP93]	• MST-based • is based on the KMB heuristic for Steiner trees • is based on Floyd's shortest path algorithm [Flo62]	$O(\Delta n^3)$ where Δ is the delay constraint.
CDKS [Sun95]	• SP-based • is based on Dijkstra's shortest path algorithm	$O(n^2)$
CKMB [Sun98]	• MST-based • is based on the KMB heuristic for Steiner trees • uses Dijkstra's shortest path algorithm	$O(mn^2)$
CAO [Wid94]	• SP-based • is based on the TM heuristic • is based on Bellman-Ford's shortest path algorithm [CLR92]	Exponential
BSMA [ZPG95]	• SP-based • is based on Dijkstra's shortest path algorithm • is based on a *k*-shortest path algorithm	$O(kn^3 \log n)$ where k is the parameter used in a *k*-shortest path algorithm
DCMA [ZKC01]	• SP-based • Least cost path and least delay path are known	$O(m \log m + n)$

As discussed in the unconstrained multicast routing algorithms, the centralized delay-constrained multicast routing algorithms are not practical for large networks. Some *distributed* delay-constrained multicast routing algorithms have been proposed, which are

not required to maintain the entire network status information on each node, and multiple nodes are participating in constructing the multicast tree, such as DKPP [KPP96][KPP93a] and DSPH [Jia98]. DKPP is based on a distributed MST algorithm with extra delay-constraint checking, while DSPH is based on the shortest path approach to expand the tree to cover each destination node one by one, similar to the TM heuristic for the Steiner tree problem, but with additional delay-constraint checking to ensure that only delay constrained shortest paths are included in the tree. As DKPP and DSPH both have been discussed in the overview and will be discussed extensively later, we do not discuss them here in more details. Table 1.3 summarizes the distributed delay-constrained multicast routing algorithms. This table also includes our published contributions which will be discussed extensively later.

Table 1.3 Distributed delay-constrained multicast routing algorithms

Algorithm	**Features**	**Message complexity**	**Time complexity**
DKPP [KPP96] [KPP93a]	• MST-based • is based on the Prim's MST algorithm • is based on Gallager, Humblet and Spira's distributed MST algorithm [GHS83]	$O(n^3)$	$O(n^3)$
DSPH [Jia98]	• SP-based • is based on the TM heuristic	$O(mn)$	$O(mn)$
ADMH [UZ01]	• MST-based • is based on the Prim's MST algorithm • handles the changes of the topology of the network	$O(n^3)$	$O(n^3)$
ASPH [UZ02a]	• SP-based • is based on the TM heuristic • handles the changes of the topology of the network	$O(mn)$	$O(mn)$
DCSP [UZ02b]	• SP-based • is based on the TM heuristic • solves the multicast routing problem efficiently	$O(mn)$	$O(n)$

The single source delay constrained multicast routing problem can be expanded to the delay constrained *group multicast routing problem* [LS02] which finds a set of constrained Steiner trees for each member in a multicast group.

In addition to the end-to-end delay constraint, multicast routing algorithms which take other QoS requirements or network resource constraints into consideration have been proposed. Rouskas and Baldine [RB97] proposed a multicast routing algorithm which considers both end-to-end delay and delay variation constraints. Bauer and Varma [BV95] proposed a multicast routing algorithm to build a minimum cost tree with node degree constrained. Given degree constraint d, then each node can only duplicate d - 1 copies of its received data.

There are also QoS requirements such as delay jitter, loss probability and bandwidth which have been considered important for QoS-based routing for both multicast and unicast [WC96]. The QoS requirements that are used for routing can be called *metrics*. According to how the metric of a route path is calculated from that of each individual link in the path, metrics are classified into *additive*, *multiplicative* and *concave* metrics if the path metric is the sum, multiplication and minimum of the metric on each individual link in the path, respectively. It is also an important topic how to select and compose metrics such that the generated multicast trees can meet more than one QoS requirements [VPL98].

1.3.3 Dynamic Steiner tree and dynamic multicast routing algorithm

The multicast routing problem modeled as Steiner tree problem assumes that the multicast group is static. In many cases, the multicast group can be dynamic, that is, new destination nodes (i.e., group members) may be added to the group, and existing destination nodes

may be removed from the multicast group. For example, in a live audio or video news conference, users may frequently join in or leave. The multicast routing problem that considers a dynamic multicast group is called *dynamic* (or *on-line*) *multicast routing problem*. For the sake of clarity, the multicast routing problem assuming a static multicast group is sometimes explicitly called *static multicast routing problem*. The dynamic multicast routing problem can be modeled as a *dynamic Steiner tree problem*, which tries to construct a sequence of minimum cost trees given a sequence of destination node addition and removal requests [Wax88][IW91].

The dynamic Steiner tree problem (or dynamic multicast routing problem) can be formulated as follows. Given a network $G=(V, E)$ with n nodes, a source node s ($s \in V$), a set of m destination nodes S ($S \subseteq V - s$) and a sequence of requests $R = \{ r_1, \ldots , r_p \}$ where each r_i is a pair (v_i, d_i) ($v_i \in V$, $d_i \in \{add, remove\}$), find a sequence of trees T_i ($T_i \subseteq G$) such that for each T_i, it is rooted at s, spans the nodes in S_i where S_i is the multicast group after satisfying requests $r_1, \ldots , r_i$, and the tree cost $\sum_{e \in T_i} C(e)$ is minimized.

Furthermore, if a delay constraint Δ is given, and for each node v in S_i, the delay on the path from s to v is bounded above by Δ; that is, $\sum_{e \in P(s, v)} D(e) < \Delta$ where $P(s, v)$ is the unique path in T_i from s to v, then the problem is called *delay-constrained dynamic multicast routing problem*.

In addition, the changes between successive multicast trees should be minimized, as data packets are constantly flowing in the multicast tree and too many changes to the tree might cause an unacceptable disruption in the packet flow. According to whether rearrangements of trees are allowed or not, dynamic multicast routing algorithms can be classified into two types: those with rearrangements and those without rearrangements.

In [Wax88][DL93], unconstrained dynamic multicast routing algorithms with no rearrangements allowed have been discussed. The difference between the greedy algorithm [Wax88] and the naive algorithm [DL93] is that the greedy algorithm tries to include a new destination node by a shortest path from the existing tree to the new destination node, while the naive algorithm connects a new destination node by a shortest path from the source to the new destination node. On the other hand, the known unconstrained dynamic multicast routing algorithms with rearrangements allowed include edge-bounded algorithm (EBA) [IW91] and ARIES (A Rearrangeable Inexpensive Edge-based on-line Steiner algorithm) [BV97].

Algorithms for the delay-constrained dynamic multicast routing problem with rearrangements have been proposed in the literature. Some are centralized algorithms with rearrangements [HLP98][SMM99], while others are distributed algorithms without rearrangements [Jia98][Sun99]. The algorithm in [SMM99] aims at satisfying the delay-constraints of all current group members while minimizing the cost of the constructed tree, and at the same time minimizing the tree changes in the multicast tree after each update request. It uses a delay-constrained shortest path algorithm to add new destination nodes to the existing multicast tree, and marks the deleted destination nodes if they cannot be removed from the tree due to the pass-through multicast traffic. After many destination nodes are deleted from the multicast group, the tree cost may be far from optimal. The algorithm uses a concept called *Quality Factor* (*QF*) (i.e., the ratio between the number of participating members and the number of deleted members) to represent the usefulness of a portion of the multicast tree to the overall multicast session. The multicast tree will be modified by a rearrangement technique only when the usefulness of a particular portion of

the tree drops below a threshold to achieve a good trade-off between the tree cost optimality and the tree stability.

Upon receiving a new destination addition request by the source node of the multicast tree, Jia's algorithm (called *Distributed Dynamic Shortest Path heuristic* (DDSP)) [Jia98] propagates this request along the existing tree to query all nodes in the tree to decide the best tree node to be used to connect with the new destination node. As soon as the best tree node is determined, the multicast tree is expanded with a delay constrained shortest path from the selected tree node to the new destination. Sun's algorithm [Sun99] is receiver(destination)-initiated and based on the multiple-path approach. The algorithm will generate multiple paths between the existing multicast tree and a receiver, and then the receiver will pick the best one based on the minimum cost to connect itself into the tree.

In addition to the dynamic group membership, the dynamic multicast routing environment can also mean the network topology changes [Ram00]. For example, a new node is added, an existing node or link fails, or the cost values associated with links change. There are many research results for the topology change problem in the unicast routing but few have been reported for the multicast routing problem. To distinguish the network topology dynamics from the group membership dynamics, we call the multicast routing problem that take the topology changes into consideration as *topology dynamic multicast routing problem.* Accordingly, we call the unicast routing that take the topology changes into consideration as *topology dynamic (unicast) routing.*

There are several reasons which cause that few multicast routing algorithms have been proposed to take care of topology changes in the network. First, it has been assumed

that today's networks are very reliable and it is not a major concern to rebuild the whole multicast tree for occasional topology changes such as node failures. Second, it has been thought that group dynamics are more important than topology dynamics as the group dynamics are unique to multicast routing problem. However, in some cases, those reasons no longer hold. For example, in host-based multicast routing, hosts could be down frequently. In these cases, topology dynamics are as important as group dynamics and even more important since topology changes in these cases are also unique to multicast routing. If they are not handled properly, it is not only an issue of optimality on resource usage and QoS requirements satisfaction but also an issue of the interruption of services.

In [NST99], a dynamic SPT algorithm for the topology dynamic routing was proposed to address the SPT problem following changes in the link states of the network. Re-computation of an entire SPT is not only inefficient but also causes frequent unnecessary changes in the topology of an existing SPT and creates routing instability. The algorithm avoids re-computing an SPT from scratch following changes in the link states of the network, by modeling the SPT problem as a dual linear programming problem so that only the affected part of an SPT may be changed.

1.3.4 Multicast routing protocol

Most multicast routing protocols adopt a multicast routing algorithm based on criteria such as correctness (i.e., loop-free routing routes), connectivity, simplicity and scalability, rather than optimality and QoS. The multicast trees constructed by them may neither be optimal on resource usage nor be satisfactory for QoS requirements. With its simplicity, the SPT algorithms have been widely used in multicast routing protocols. The SPT algorithms optimize the path to each individual destination but may not optimize the cost

of a multicast tree as a whole. Often, the path optimization is on the delay or number of hops in a path. MOSPF [Moy94] uses Dijkstra's shortest path algorithm [Dij59] to construct a *source-specific* (i.e., a tree that is rooted at the source node) SPT as a multicast tree. Distance Vector Multicast Routing Protocol (DVMRP) [Dee88] and Protocol Independent Multicast - Dense Mode (PIM-DM) [Dee97] use the reverse path forwarding (RPF) algorithm [DM78] to construct a source-specific SPT as a multicast tree. The RPF algorithm assumes that the costs associated with links are symmetric, so it expands an SPT along the reverse shortest path between the source and each destination. However, in many cases, the costs associated with links are asymmetric. So, the multicast routing tree is not optimal. But the asymmetric problem and non-optimal multicast routing tree seems not to bother the designers of multicast routing protocols. The major reason probably is that the RPF method guarantees that the generated multicast tree is loop-free. In addition, the RPF method reuses existing unicast routing state information at each node and it can be easily implemented in a distributed form. EXPRESS [HC99] uses the receiver-initiated approach to create a reverse SPT for multicast routing.

In addition to source-specific trees, multicast routing protocols also use a *shared tree* as a multicast tree, which is a tree rooted at a *core* node which is not necessarily a source. Both senders (i.e., source nodes) and receivers (i.e., destination nodes) connects to the tree via a shortest path towards the core. Multicast protocols Core Based Trees (CBT) [BFC93], Protocol Independent Multicast - Sparse Mode (PIM-SM) [Dee96], Multicast Internet Protocol (MIP) [PG97], Border Gateway Multicast Protocol (BGMP) [Kum98], and Simple Multicast [Perl99] use a shared tree for multicast routing. Obviously, the paths from a source to receivers in a shared tree are not optimal. Using

shared trees are more scalable than using source-specific trees since shared trees require a single routing entry per group while source-specific trees require a routing entry per source per group. Considering the implementation issues such as the size of the routing table and the number of required timers, the shared tree based protocols should be favored [Bil97]. However, the work in [WE94] indicates that it would be ideal to flexibly support source-specific trees as well as shared trees within one multicast architecture since the performance provided by an SPT such as minimum delay may not be achievable by a shared tree. So, if the multicast traffic is high enough to warrant a source-specific tree or path, PIM-SM and MIP can switch to a source-specific tree, while BGMP can switch to a source-specific branch.

In shared trees, it may not be the source who initiates the construction of a multicast tree. It can be the destinations (or receivers) to initiate the construction of a reverse shortest path between the core and themselves. This approach has also been used in source-specific trees where new receivers initiate the construction of a reverse shortest path between the source and itself. So, these protocols are also called *receiver-initiated multicast routing protocols* to differentiate them from the *source-initiated multicast routing protocols*. Both CBT and PIM-SM are receiver-initiated multicast routing protocols.

Recently some QoS-based multicast routing protocols such as YAM [CC97] and QoS sensitive Multicast Internet protocol (QoSMIC) [FBP98] have been proposed. Both protocols construct shared trees. However, instead of using the reverse shortest path between the core and a group member for the member to join the multicast tree, these protocols offer multiple paths to a new group member, who selects the best path among

the alternate paths according to the defined QoS requirements such as end-to-end delay or bandwidth. Based on whether a single path or multiple candidate paths are provided for a new member to join a multicast tree, the multicast routing protocols can be classified into *single-path* multicast routing protocols and *multiple-path* multicast routing protocols. The multiple-path approach increases the chance of finding a QoS satisfied multicast tree.

In YAM, a new member broadcasts join-request messages in its neighborhood to find on-tree nodes, which is called *spanning join*. Each on-tree node that receives the message sends a reply message back to the new member. The reply message carries a candidate path which is the path that the reply message travels. It also collects the QoS properties of the path as it traverses. The new member selects the best path among all received candidate paths. This search procedure may be repeated with increased search scope until some on-tree nodes are found.

QoSMIC starts with a shared tree and individual receivers switch to a QoS-competitive source-specific tree when necessary. When a new member wants to join the group, two procedures called *local search* and *multicast tree search* are used to derive the multiple candidate paths between the existing multicast tree and the new member. In the local search, the new member floods a path probe message to its neighborhood with a limited scope. Those on-tree nodes that receive the probe message will respond with a candidate path. In the multicast tree search, some on-tree nodes selected by a special *Manager* node will send a candidate path to the new member. Each candidate path message calculates the concerned QoS metrics along its way to the new member. Eventually, the new member compares all received alternate paths and choose the best path to join the multicast tree according to the QoS requirements. Depending on the

applications, the best path could be minimum end-to-end delay path or maximum bandwidth path or something else. It can be seen that the local search procedure is equivalent to the search procedure used in YAM, but with the multicast tree search, QoSMIC can restrict its local search in a small neighborhood.

The extra control message overhead could be very high in these protocols as they are based on some flooding mechanism to find a QoS satisfied path for each receiver. Though QoSMIC tries to reduce such overhead by controlling the flooding scope, it does not eliminate the flooding behavior. The QoS-aware Multicast Routing Protocol (QMRP) proposed in [CNS00] goes further by avoiding the costly multiple path search as much as possible. It starts with a single path searching but can expand the search to multiple paths by splitting at one or multiple points of the initial path when the initial path does not satisfy the QoS requirements. The points for splitting are selected according to the current network conditions. On the other hand, the Receiver-Initiated Multicast protocol with multiple QoS constraints (RIMQoS) [FG00] uses QoS-based unicast protocol to construct a single QoS path which connects a new member with the existing multicast tree.

Table 1.4 summarizes the comparison among all discussed multicast routing protocols with respect to their multicast routing algorithm, type of tree, initiation of the tree construction, and QoS awareness. There are good surveys on multicast routing protocols which are from different perspectives and have different focuses [Ram00] [WH00][CN98][DDC97][Pau98].

Table 1.4 Multicast routing protocols

Protocol	**Multicast routing algorithm**	**Tree type**	**Initiation of tree construction**	**QoS-based**
DVMRP	• Reverse SPT	source-specific	source-initiated	No

MOSPF	• SPT based on Dijkstra's shortest path algorithm	source-specific	source-initiated	No
PIM-DM	• Reverse SPT	source-specific	source-initiated	No
PIM-SM	• SPT	shared and source-specific	receiver-initiated	No
CBT	• SPT	shared	receiver-initiated	No
MIP	• SPT	shared and source-specific	receiver-initiated	No
EXPRESS	• Reverse SPT	source-specific	receiver-initiated	No
YAM	• multiple-path	shared	receiver-initiated	Yes
QoSMIC	• multiple-path	shared and source-specific	receiver-initiated	Yes
QMRP	• constrained SP and multiple-path	shared	receiver-initiated	Yes

1.3.5 Simulated network topology

To evaluate the performance of various multicast routing algorithms, two types of simulated random network topologies, namely Waxman model [Wax88] and Tiers model [Doa96][CDZ97], are adopted. Waxman model has been used extensively for algorithm evaluations in communication networks. Tiers model has been known as a random graph model to approximate the hierarchical structure of the Internet network topology. In our study, we choose to use Waxman model for the following reasons:

- Waxman model has been used much more than the other one;
- Using Waxman model makes our work comparable to the work of others;
- The algorithms that we studied do not assume that the network has the hierarchical structure of the Internet network topology. So, we should use a generic network topology model like Waxman's.

1.4 Network simulator for performance evaluation - Waxman's approach

In order to compare the performance of different algorithms, a series of simulations are performed by applying all algorithms under consideration to networks generated by the Waxman's approach [Wax88]: the nodes are randomly distributed over a rectangular coordinate grid. Each node is placed at a location with integer coordinates. The Euclidean metric is then used to determine the distance between each pair of nodes. A link between two nodes u and v is added with a probability that is given by the function $P(u, v) = \beta \exp(-d(u, v)/ \alpha L)$, where $d(u, v)$ is the distance from u to v, L is the maximum distance between any two nodes, and $0 < \alpha \leq 1$, $0 < \beta \leq 1$. Larger values of β result in graphs with higher link densities, while small values of α increase the density of short links relative to longer ones. The cost of a link is assigned to a value which is uniformly distributed over the range between 0 and 60. The delay of a link (u, v) in the graph is the distance between nodes u and v on the rectangular coordinate grid. Graphs are generated and tested until the graph is a two-connected network, which has at least two paths between any pair of nodes. The random graphs do have some of the characteristics of an actual network. It has been shown by simulation [NT94] that the performance of a multicast routing algorithm when applied to a real network is almost identical to its performance when applied to a random two-connected network. The size of messages is not taken into consideration in performance measurement.

1.5 Scope, objectives and contributions

It can be seen that the trends in developing multicast routing algorithms are towards distributed QoS-based multicast routing algorithms. Meanwhile, it can also be seen that

few multicast routing algorithms are adopted in multicast routing protocols due to their different design objectives.

It has been observed that there are many research results for the topology change problem in the unicast routing but few have been reported for the multicast routing problem. People tend to believe that today's networks are very reliable and it is not a major concern to rebuild the whole multicast tree for occasional topology changes such as node failures. Also, they believe that group dynamics are more important than topology dynamics as the group dynamics are unique to multicast routing problem. We have indicated that in some cases such as host-based multicast routing, topology dynamics are as important as group dynamics and even more important than group dynamics since topology changes in these cases are frequent and unique to multicast routing. If they are not handled properly, it is not only an issue of optimality on resource usage and QoS requirements satisfaction but also an issue of the interruption of services.

It has also been observed that the existing distributed QoS-based multicast routing algorithms and protocols try to construct a multicast tree by following the approach of adding destination nodes one by one, which has a very low efficiency.

Furthermore, it has been observed that although the group dynamics in multicast routing has been addressed in some researches, the existing distributed QoS-based multicast routing algorithms use the approach such as querying the whole or a part of a multicast tree before a new destination node is added, which has a very low efficiency.

Based on the issues that we observed and the trends in the development of multicast routing algorithms and protocols (i.e., distribution and QoS-based), we focus our work on designing new distributed QoS-based multicast routing algorithms which are

more reliable and efficient than the existing ones. We hope that the results could bridge the gap between the multicast routing algorithms and protocols.

Our contributions for solving the delay constrained multicast routing problem start with the following two delay constrained multicast routing algorithms considering the dynamic network topology changes:

- a new distributed delay constrained multicast routing algorithm (called *Adaptive distributed Shortest Path Heuristic* (ASPH)) that is SP-based and capable of doing a local fault recovery from node failures in the network. The performance evaluation shows that the proposed algorithm performs better in terms of message and time complexity than the leading SP-based distributed delay constrained multicast routing algorithm called *Distributed Shortest Path Heuristic* (DSPH) [Jia98] which has to conduct a complete fault recovery from node failures in the network.
- a new distributed delay constrained multicast routing algorithm (called *Adaptive Distributed MST-based Heuristic* (ADMH)) that is MST-based and capable of fault recovery from node failures in the network. The performance evaluation shows that the proposed algorithm performs better in terms of message and time complexity than the leading MST-based distributed delay constrained multicast routing algorithm called *Distributed KPP heuristic algorithm* (DKPP) [KPP96] which has to conduct a complete fault recovery from node failures in the network.

The next contribution presented in this thesis is a new distributed delay constrained multicast routing algorithm (called *Distributed Concurrent Shortest Path heuristic* (DCSP)) with message complexity O(mn) and time complexity O(n) that is more efficient than any existing distributed delay constrained multicast routing algorithms in

terms of time complexity. The efficiency of the proposed algorithm is achieved by covering all destination nodes concurrently instead of covering each destination node sequentially as in the traditional distributed delay constrained multicast routing algorithms.

The remaining contributions are two distributed delay constrained multicast routing algorithms that consider the dynamic network topology and group membership changes:

- a new distributed delay constrained multicast routing algorithm (called *Adaptive distributed Concurrent Shortest Path heuristic* (ACSP)) that augments our proposed concurrent distributed delay constrained multicast routing algorithm DCSP and capable of doing a local fault recovery from node failures in the network. The performance evaluation shows that the proposed algorithm performs better in terms of message and time complexity than DCSP algorithm which has to conduct a complete fault recovery from node failures in the network.
- a new distributed delay constrained dynamic multicast routing algorithm (called *Distributed Dynamic Concurrent Shortest path heuristic* (DDCS)) that expands our proposed concurrent distributed delay constrained multicast routing algorithm DCSP and capable of handling the addition and deletion of destination nodes in the multicast group. The performance evaluation shows that the proposed algorithm performs better in terms of message and time complexity than the leading distributed delay constrained dynamic multicast routing algorithm called *Distributed Dynamic Shortest Path heuristic* (DDSP) [Jia98]. The results show that the number of messages required by DDSP is 5-20 times as many as that required by DDCS, and the convergence time

required by DDSP is 2-3 times as many as that by DDCS during the addition of a new destination node.

1.6 Organization of the manuscript

The remainder of this manuscript is organized as follows. Chapter 2 presents the Adaptive distributed Shortest Path Heuristic algorithm ASPH that is SP-based and capable of achieving local fault recovery from node failures in the network. Then it discusses the performance evaluation conducted on the proposed algorithm. Chapter 3 describes the Adaptive Distributed MST-based Heuristic algorithm ADMH that is MST-based and capable of fault recovery from node failures in the network. The performance evaluation of the proposed algorithm is presented after the description of the algorithm. Chapter 4 discusses the Distributed Concurrent Shortest Path heuristic algorithm DCSP that achieves a high efficiency through covering all destination nodes concurrently instead of covering each destination node sequentially as in the existing distributed delay constrained multicast routing algorithms. Chapter 5 augments the Distributed Concurrent Shortest Path heuristic algorithm DCSP with a fault recovery approach to recover from node failures in the network effectively and efficiently. The resulting augmented algorithm is called Adaptive distributed Concurrent Shortest Path heuristic ACSP. Chapter 6 presents the Distributed Dynamic Concurrent Shortest path heuristic algorithm DDCS that is capable of handling the addition and deletion of destination nodes in the multicast group effectively and efficiently. Then, it discusses the performance evaluation conducted on the proposed algorithm. Finally, Chapter 7 provides a summary of the contributions and the final remarks.

Chapter 2

QoS-based Multicast Routing with Fault Recovery for Real-Time Communication

2.1 Introduction

One of the main problems with the distributed delay constrained multicast routing algorithms is that they do not take into account the changes in the topology of the network (e.g., node failures) [BV96]. Whenever the topology of the network changes, these algorithms will fail to complete the construction of a multicast tree and it is up to the application to re-start the algorithm. As a result, these algorithms will have a low success rate for the construction of a multicast tree (which is the number of trials to build a multicast tree successfully divided by the number of total trials). Similarly, as these algorithms do not respond to the changes in the topology of the network after the multicast tree is built, it is up to the application to re-start the algorithm to re-build the tree for the changed network topology, which causes the interruption of the running traffic for all existing members in a multicast session. The re-starting approach can be broadly considered as a kind of fault recovery approach which will be called *naïve fault recovery approach.*

In this chapter, we propose a new distributed delay constrained multicast routing algorithm that takes into consideration the node failures in a network and recovers from these failures. Though we only consider the node failures in the proposed algorithm, the

failures. The proposed algorithm will recover by itself and adaptively construct a constrained multicast tree when node failures occur. So, the proposed algorithm is called *Adaptive distributed Shortest Path Heuristic* (ASPH).

In addition, we want ASPH to generate the delay constrained multicast trees whose quality is as good as that of DSPH in terms of tree cost, and perform as well as DSPH in terms of message complexity. These goals are in general conflicting with each other. For example, to be able to do fault recovery, extra messages will be needed. The key approach used in ASPH is to localize the recovery action to the failed portion of the multicast tree without re-building the whole tree again. Then, the problem is how the information about the failed sub-tree can be communicated to the active node, so that the new information can be taken into consideration in the rest of the tree construction process. One straightforward approach is to flood the information in the network. However, this approach requires $O(n^2)$ messages just for the notification of the node failure alone. We do not take this approach. In ASPH, the failed sub-tree is refrained from flooding the network with fault information messages, and the fault information is propagated through the regular tree setup messages as much as possible.

With respect to the main control steps of the algorithm, ASPH progresses in the same way as DSPH. Between the steps, ASPH checks if any tree node fails. When a failure is detected, it removes the sub-tree rooted at the failed node and notifies the source node s of the destination nodes that were covered by the removed sub-tree. Thus, the source node s is enabled to add these removed destination nodes later as ASPH will report back to the source node s when all the remaining destination nodes have been added to the multicast tree.

Furthermore, loops may be introduced in the multicast tree due to the network topology changes. ASPH will detect these loops by checking if a node (say v) to be added is already in the tree. A loop is removed by choosing the path from the source node s to node v which has the minimum delay. This will ensure that all existing paths between the source node s and destinations that go through node v still satisfy the delay constraint. If the new parent of v has the minimum delay from the source node, node v will accept the new parent and break itself from its previous parent. Otherwise, node v will reject the new parent.

After the tree is constructed, ASPH will continue to run the tree node failure checking and recovery steps to repair the delay constrained multicast tree if node failures occur during an ongoing multicast session.

2.2.2 Details of the proposed SP-based algorithm

Eight types of messages are used in ASPH, which are

open - opening a multicast connection;

setup - setting up a shortest path from the tree to a non-tree node;

fork - forking a new branch from the tree node that is closest to the selected non-tree node;

completion - notifying the termination of the multicast tree setup;

break - notifying its parent to break a loop;

reject - rejecting the invitation to join the tree as either the constraint may be violated or a loop may be formed;

remove - removing a sub-tree from the tree;

destination - adding destination back to the uncovered destination list;

Among these messages described above, the first four types of messages are used in DSPH as well, while the last four types of messages are newly introduced for ASPH. It is assumed that node failures are detected by a lower level protocol. The pseudocode of ASPH is shown in Figure 2.1.

Figure 2.1: Pseudo-code of ASPH Algorithm

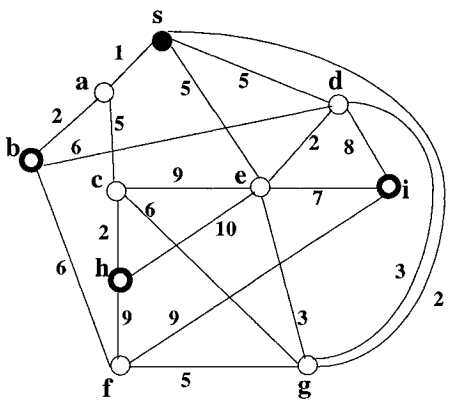
```
/* local = local node                                  */
variables:
/* s = source node                                     */
/* D = accumulated delay from the source               */
/* L = tree level number from the source               */
/* T2D = tree to destination table                     */
/*    tree_node - tree node;                           */
/*    cost - distance between tree and the             */
/*           destination                               */
/*    tag - flag to indicate if the destination        */
/*          has been covered                           */
/* Route = route table, which has                      */
/*    next - next hop node                             */
/*    cost - the distance to a destination             */
/*    delay - the delay to a destination               */
/* dest = destination node                             */
/* sync_list = destination list needs to be            */
/*             synchronized                            */

initialize:
D := 0;
L := 0;


/* Transaction 1: */
on receiving open(S, Δ):
  s := local;
  sync_list := nil;
  add s to the tree;
  /* Transaction 1.1: initialize T2D */
  for each d_i ∈ S do
    T2D(d_i).cost := Route(d_i).cost;
    T2D(d_i).tree_node := s;
    T2D(d_i).tag := no;
  end
  /* Transaction 1.2: select the first destination
     to be covered */
  find d_i: T2D(d_i).tag = no ∧ ∀ d_j ∈ S,
      T2D(d_i).cost ≤ T2D(d_j).cost
  T2D(d_i).tag := yes;
  dest := d_i;
  send setup(dest, T2D, D, L, s, S, Δ) to
       Route(d_i).next;

/* Transaction 2: */
on receiving setup(dest, T2D, D, L, s, S, Δ)
  T2D := setup.T2D;
  dest := setup.dest;
  do_fork := false;
  /* Transaction 2.1: loop checking */
  if node local is already in the tree then
    /* 2.1.1: reject the setup request */
    if D + Route(dest).delay < Δ then
      T2D(dest).cost := Route(dest).cost;
      T2D(dest).tree_node := local;
      send reject() to sender;
    /* 2.1.2: accept the new setup request and
       break from the existing tree */
    else if setup.L < L and setup.D < D then
      send break() to parent;
      add node local to the tree;
      L := setup.L + 1;
      D := setup.D + delay(sender, local);
    /* 2.1.3: the delay bound may be violated
       and set source as the tree node for dest */
    else
      T2D(dest).tree_node := s;
      T2D(dest).tag := no;
      T2D(dest).cost := ∞;
      do_fork := true;
    endif
```

```
  else /* It is OK to add node local to the tree */
    add node local to the tree;
    D := setup.D + delay(sender, local);
    L := setup.L + 1;
    /* modify T2D table */
    for each d_k ∈ S and T2D(d_k).tag = no do
      if (D + Route(d_k).delay ≤ Δ) and
        Route(d_k).cost < T2D(d_k).cost then
        T2D(d_k).cost := Route(d_k).cost;
        T2D(d_k).tree_node := local;
      endif
    end
  endif

  /* Transaction 2.2: not destination yet */
  if local ≠ dest and not do_fork then
    send setup(dest, T2D, D, L, s, S, Δ) to
      Route(dest).next;
  /* Transaction 2.3: all destinations are
    covered */
  else if all destinations are covered then
    send complete() to s;
  /* Transaction 2.4: fork to cover next
    destination */
  else
    find d_i: T2D(d_i).tag = no ∧ ∀ d_j ∈ S,
          T2D(d_i).cost ≤ T2D(d_j).cost
    dest := d_i;
    send fork(dest, T2D) to T2D(d_i).tree_node;
  endif

/* Transaction 3: */
on receiving fork(dest, T2D)
  T2D := fork.T2D;
  dest := fork.dest;
  if node local is in the tree then
    find d_i: T2D(d_i).tag = no ∧ ∀ d_j ∈ S,
          T2D(d_i).cost ≤ T2D(d_j).cost
    T2D(d_i).tag := yes;
    dest := d_i;
    send setup(dest, T2D, D, L, s, S, Δ) to
        Route(d_i).next;
  else
    /* re-fork */
    T2D(dest).tree_node := s;
    T2D(dest).tag := no;
    T2D(dest).cost := ∞;
    execute Transaction 2.4;
  endif

/* Transaction 4: */
on detecting the failure of a child node
  remove the child node from the tree;

/* Transaction 5: */
on detecting the failure of its parent node or
    receiving remove()
  remove local from the tree;
  if local is not a leaf node then
    for each child in the tree do
      send remove() to child;
    end
  else if local is a destination node then
    send destination() to s;
  endif

/* Transaction 6: */
on receiving destination()
  add sender to sync_list

/* Transaction 7: */
on receiving complete()
  if sync_list ≠ nil then
    /* Add nodes in sync_list back to the list of
      destinations to be covered */
    for each d_i ∈ sync_list do
      T2D(d_i).cost := Route(d_i).cost;
      T2D(d_i).tree_node := s;
      T2D(d_i).tag := no;
    end
    sync_list := nil;
    /* fork to cover next destination */
    execute Transaction 2.4;
  else
    announce that the tree is ready for use;
  endif
```

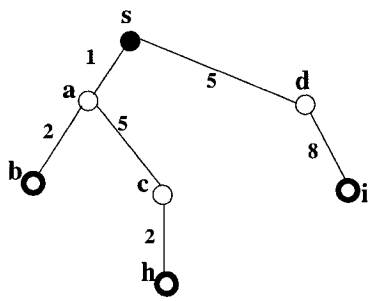
Interleaved with the detailed description of the proposed algorithm, an example network shown in Figure 2.2 is used to illustrate the algorithm. For clarity of the diagrams, the same integer number is used to represent both cost and delay values. Node *s*

is the source node and dark nodes *b*, *h* and *i* are destination nodes. Δ is 14 and node *e* fails after node *b* and *h* are covered.

Figure 2.2 An example for ASPH Heuristic



(a) a network



(b) a multicast tree for (a)

dest	*cost*	*tree_node*	*tag*
b	3	s	no
h	8	s	no
i	12	s	no

(c) initial T2D table

dest	*cost*	*tree_node*	*tag*
b	3	s	yes
h	7	a	no
i	12	s	no

(d) T2D table updated by node *a*

The details of the proposed algorithm are as follows:

1. When a node receives a request (*open* message) for opening a multicast connection with parameters S and Δ, it becomes the source node s of the multicast connection (i.e., the only tree node in the multicast tree).

   In Figure 2.2, when node s receives an *open* message, it becomes the first node in the tree.

   1.1 The source node then calculates an initial T2D (tree to destination) table. For each destination $d_i \in S$, T2D table records the following information: *cost* - the cost from the tree to d_i, *tree_node* - the tree node closest to d_i and *tag* - indicating if d_i is in the tree or not. Obviously, the *cost*, *tree_node* and *tag* fields for each destination in

T2D table will be initially set to the distance of the shortest path from the source node s to the destination, source node s and "*no*", respectively.

1.2 The destination closest to the tree is selected and its *tag* in T2D table is marked as "*yes*". A *setup* message is sent to the neighbor towards the selected destination to include all nodes on the shortest path into the tree. This setup message carries T2D table and the accumulated delay and the number of hops (called *tree level*) from the source node s.

In Figure 2.2, node b will be selected as the closest destination based on Figure 2.2(c).

2. When a node v receives a *setup* message, it includes itself into the tree and modifies T2D table if a lower cost from itself to a destination is found, the destination is not yet in the tree and the delay from itself to the destination plus the accumulated delay from the source node s is under Δ.

In Figure 2.2, when node a receives the setup message, node a will update T2D table as in Figure 2.2(d).

2.1 If the addition of node v to the tree will introduce a loop (which means that node v is already in the tree), node v will do the following to avoid the loop:

2.1.1 If the accumulated delay along the existing path from the source node s to the destination node is within the delay bound, then node v sends a *reject* message to the sender of the setup message. So, the sender won't include node v in the tree.

2.1.2 Else if the sender node has higher or equal tree level than node v and the new accumulated delay from the source node s to node v via the sender is less

than the old accumulated delay, then node v sends a *break* message to its parent to break the existing tree path and accept the new one.

2.1.3 Otherwise, field *tree_node* in T2D for the destination under consideration will be set to the source node s to force the algorithm to re-build the tree path to the destination from the source node s.

2.2 If node v is not the destination yet, node v simply passes the *setup* message to the neighbor towards the destination with the possibly modified T2D table and the adjusted accumulated delay from the source node s.

2.3 If all destinations are included in the tree (i.e., all tags in T2D table are "*yes*"), node v sends a *completion* message to the source node s.

2.4 If node v is the destination itself, it selects next closest destination and sends a *fork* message to *tree_node* that is recorded in T2D table. The *fork* message carries T2D table.

In Figure 2.2, node b will send a *fork* message to node a to include the next selected destination h.

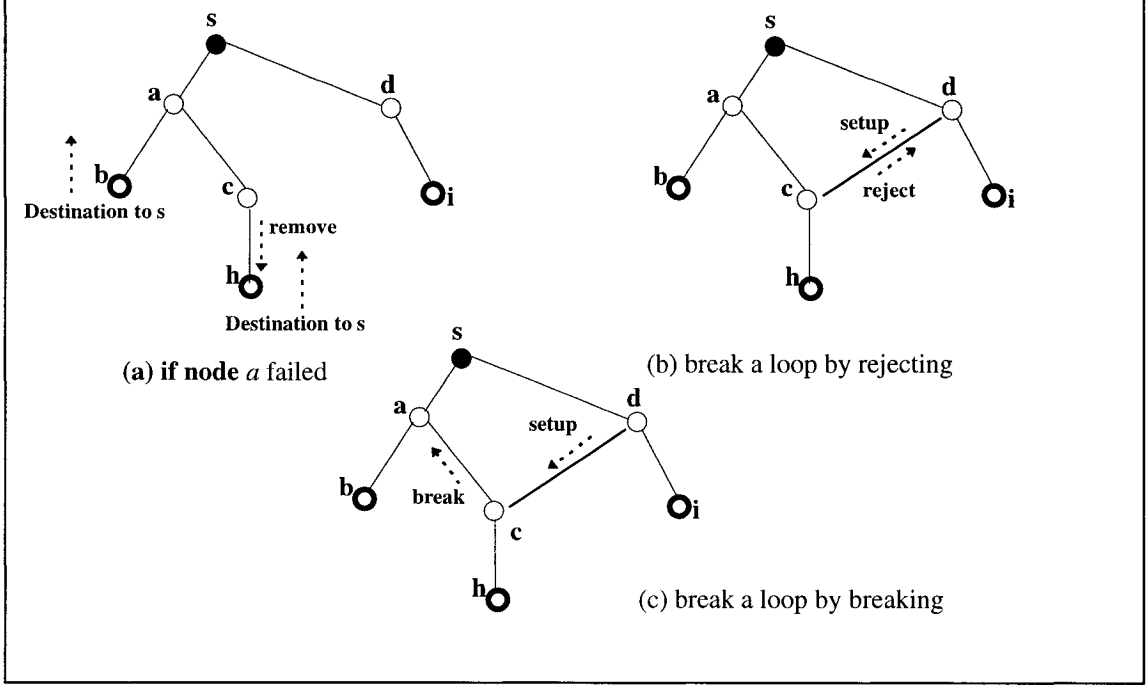
3. When node v receives the *fork* message, if node v is in the tree, the destination closest to the tree is selected and its *tag* in T2D table is marked as "*yes*". Node v then creates a *setup* message and forwards it to the neighbor towards the destination with T2D table and the adjusted accumulated delay and the tree level from the source node s. If node v is not in the tree, field *tree_node* in T2D for the destination under consideration will be set to the source node s to force the algorithm to re-build the tree path to the destination from the source node s. Then it re-forks as in Transaction 2.4.

The final multicast tree for the example is shown in Figure 2.2(b).

4. When node v in the tree detects that its child node fails, node v removes the child node from the tree.
5. When node v in the tree detects that its parent node fails or receives the *remove* message, node v removes itself from the tree. If node v is not a leaf node, it will forward the *remove* message to all of its children. If node v is a destination node, it sends a *destination* message to the source node s so that the source node s will add the destination node back to the uncovered destination list when the merging of the uncovered destination list occurs.
6. When the source node s receives the *destination* message, it adds the sender node to a "sync list" which is the uncovered destination list.
7. When the source node s receives the *completion* message, if "sync list" is not empty, the *cost*, *tree_node* and *tag* in T2D table for each destination $d_i \in$ "sync list" will be re-set to the distance of the shortest path from the source node s to the destination, source node s and "*no*", respectively. Then it re-forks as in Transaction 2.4. If "sync list" is empty, it announces that a delay-constrained multicast tree is ready for use.

Figure 2.3(a) illustrates the procedure described in transactions 4 to 6, where *remove* and *destination* messages are sent. Figure 2.3(b) illustrates the case described in 2.1.1, where the *setup* message is rejected to avoid loops in the tree, while Figure 2.3(c) illustrates the case described in 2.1.2, where the loop can be broken by breaking the existing tree branch.

Figure 2.3 Illustrations for ASPH Heuristic



The correctness of the proposed algorithm is shown by the following theorem.

Theorem 2.1: The delay-constrained multicast tree built by ASPH is loop free.

Proof: If there is a loop in the tree, there must be a node which has two parent nodes. In ASPH, each time a node receives a *setup* message for joining the tree, the node checks if it is already in the tree. If it is not in the tree yet, it will join the tree and set the sender of the *setup* message as its parent node. Otherwise, it will either reject the *setup* message or accept the *setup* message but break from its existing parent node first. So, in all cases, each node has one and only one parent node. Therefore, the delay-constrained multicast tree built by ASPH is loop free. EOP.

Now, let us determine whether the proposed algorithm generates a constrained multicast tree. The answer is affirmative, and it is proved by the following theorem.

Theorem 2.2: When ASPH terminates with a multicast tree, the multicast tree is constrained; that is, the tree satisfies the delay constraint.

Proof: According to the algorithm, a node v can expand the multicast tree to node w only if for the destination $d \in S$ that is going to be routed via w, $P(v) + D(v,w) + SD(w,d) < \Delta$, where $SD(w,d)$ = minimum delay of the shortest path from w to d; $D(v,w)$ = delay on the link (v,w); $P(v)$ = delay on path from s to v in the tree. So, every node that is included into the multicast tree must satisfy the delay constraint.

In case that w is already in the tree, as the algorithm allows w to become node v's child node only if the new accumulated delay value $P(w)$ is smaller than the older one, it is guaranteed that all destinations that have been chosen to be routed via w will continue to satisfy the delay constraint. Therefore, if the algorithm successfully builds a multicast tree, it will be a delay constrained one. EOP.

Next, let us discuss the performance of ASPH in terms of the number of message exchanges and the convergence time, and the quality of the generated multicast tree by ASPH in terms of the tree cost. Without node failures, DSPH runs with $O(mn)$ message and time complexity in the worst case. As each node failure will make DSPH re-run, DSPH will have a $O(kmn)$ message and time complexity if there are k-1 node failures during a multicast session. The following theorem shows that in the worst case, ASPH performs as good as DSPH in terms of the message and time complexity. The simulation conducted in the next section will complement the analysis made here by comparing ASPH with DSPH further in terms of the number of message exchanges and the convergence time in the average case. The simulation also makes the comparison between these two algorithms in terms of the tree cost.

Theorem 2.3: ASPH's message complexity is O(*kmn*) and time complexity is O(*kmn*) in the worst case, where *k*-1 is the number of node failures occurred in the network during the construction of a delay constrained multicast tree and the on-going multicast session.

Proof: The *k*-1 node failures force ASPH to try *k* times to complete the construction of a delay constrained multicast tree. Each time, the *setup* message for each destination will be sent O(*n*) times. Since there are *m* destinations, there will be O(*mn*) number of *setup* messages. The *destination* message will not be sent more than *m* times as only one *destination* message can be sent for each destination. The *break* or *reject* messages can be sent O(*n*) times along the way to each destination. The *remove* messages are sent once along each edge of the sub-tree rooted at the failed node, which is O(*n*) times. So, the algorithm runs with a message complexity of O(*kmn*).

The setup message will reach each destination in O(*n*) time for each trial of completing a delay constrained multicast tree and there are *m* destinations. So, the algorithm runs with a time complexity of O(*kmn*). EOP.

2.2.3 Modification of the proposed SP-based algorithm for link failure

In our discussion so far, we focus on the node failure among all kinds of network topology changes, since it is the most complicated case. However, an algorithm that can handle the node failure does not automatically have the capability to handle other kinds of topology changes (e.g., link failure). In order to handle the link failure as well, the algorithm has to be modified. In ASPH, when a node failure is detected, it removes the sub-tree rooted at the failed node and notifies the source node *s* of the destination nodes that were covered by the removed sub-tree. Thus, the source node *s* is enabled to add these removed destination nodes later. Similarly, the link failure can be handled as follows: when a link

failure is detected, it removes the sub-tree rooted at the end node of the failed link that is further away from the source node and notifies the source node s of the destination nodes that were covered by the removed sub-tree; thus, the source node s is enabled to add these removed destination nodes later. So, ASPH can be modified to cover the link failure case.

Only step 4 and 5 in the details of ASPH need to be replaced with the following new steps for the link failure case, and all other steps can be kept intact:

4. When node v in the tree detects that the link connecting to its child node fails, node v removes the child node from the tree.
5. When node v in the tree detects that the link connecting to its parent node fails or receives the *remove* message, node v removes itself from the tree. If node v is not a leaf node, it will forward the *remove* message to all of its children. If node v is a destination node, it sends a *destination* message to the source node s so that the source node s will add the destination node back to the uncovered destination list when the merging of the uncovered destination list occurs.

2.3 Performance analysis

In this section, we compare the performance of ASPH with DSPH in the average case under the condition that a node failure or link failure occurs during the construction of a delay constrained multicast tree or during the on-going multicast session. DSPH is going to use the naïve fault recovery approach; that is, the algorithm will be re-run from scratch when a node failure occurs. To the best of our knowledge, there are no other fault recovery approaches that have been published for the delay constrained multicast tree problem. Otherwise, we would include them in the comparison as well.

In order to compare the performance of ASPH with DSPH, a series of simulations have been performed by applying both DSPH and ASPH to networks generated by the Waxman's approach as explained in Section 1.4.

2.3.1 Node failure

In the simulations, node failures are injected into networks in order to see how the multicast routing algorithms perform in a network where node failures occur. There are two types of experiments: one is for the case when node failures occur during the construction of a multicast tree, and the other is for the case when node failures occur during the on-going multicast session. For the first type of experiments, it is assumed that at most one node failure may occur during the construction of the multicast tree. The timing for a node failure is randomly selected so it could occur randomly among the different stages of the construction of the multicast tree. The failed node is randomly selected among the nodes in the multicast tree built so far that are neither the source node nor the destination nodes when node failure occurs, since the failure of the source node means that there will be no multicast tree to be built and the failure of a destination node means that the constructed multicast tree will not be comparable with other multicast trees which cover all destination nodes, because the more nodes need to be covered, the higher the number of messages and time complexity are. DSPH will be re-run when a node failure occurs.

The number of messages exchanged, the convergence time and the cost of the multicast tree are measured by their average value in a total of 100 simulation runs on a network with 200 nodes. Note that an exchanged message will only be counted once from its sender to its receiver no matter how many intermediate nodes are walked through as

long as the algorithm running on the node does not interpret the message. Meanwhile, the convergence time is counted by taking one message exchange as a time unit. However, within one time unit, there may be several message exchanges occurring in the network. Thus, multiple messages exchanged within the same time unit will only be counted once in the convergence time.

For the second type of experiments, DSPH will have to be re-run to re-build the entire multicast tree when a node failure occurs during an ongoing multicast session. In the experiments, one multicast tree node will be randomly selected as a failure node during a multicast session for the type of networks that we study. Like the first type of experiments, the number of messages exchanged, the convergence time and the cost of the multicast tree are measured by their average value in a total of 100 simulation runs on a network with 200 nodes.

As DSPH will be re-run when a node failure occurs while ASPH will always return a multicast tree no matter whether a node failure occurs or not, ASPH has a better success rate than DSPH.

Figure 2.4, Figure 2.5 and Figure 2.6 show the simulation results when Δ is set to $d_{max} + (3/8)d_{max}$, where $d_{max} = \max(\{d_u \mid \text{for any } u \in S\text{: } d_u \text{ is the delay on the shortest path from } s \text{ to } u\})$, and the group size changes between 5 and 60 in 200-node networks. In the figures, suffix "-c" means during the construction of a multicast tree, suffix "-m" means during the on-going multicast session and "SPT-d" means the delay-based SPT. The delay-based SPT could be considered as the delay constrained multicast trees without any optimization on tree costs.

Figure 2.4: Tree cost versus group size when node failure occurs

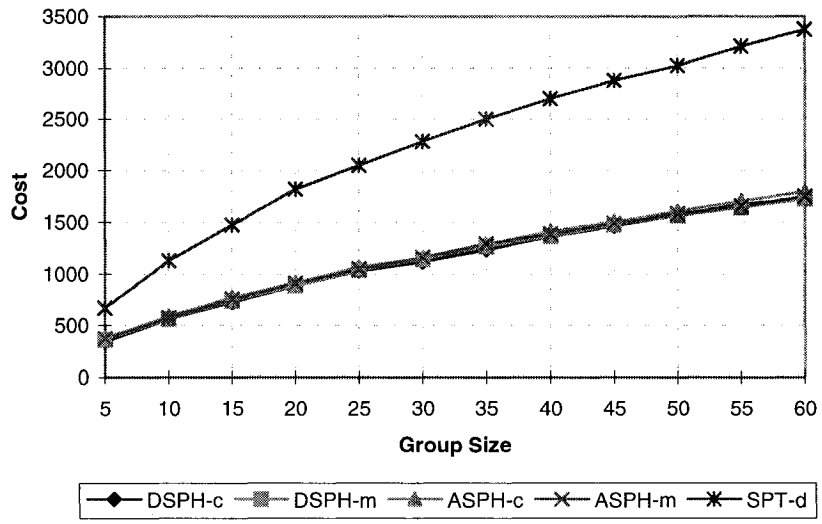


Figure 2.4 shows that the costs of the trees generated by ASPH are almost identical to those by DSPH. This result is very encouraging. DSPH calculates the multicast tree based on the consistent current network topology information after it re-runs while ASPH might use different network topology information for different parts of a delay constrained multicast tree. Intuitively, it could be expected that the costs of the trees generated by ASPH should be noticeably higher than those by DSPH. But, the simulation results show that it is not the case. It is also shown in Figure 2.4 that the delay-constrained multicast tree algorithms generate trees with much better cost performance than the algorithms without considering optimization on the tree cost such as SPT-d. This means that it is worth using the delay-constrained multicast tree algorithms rather than using SPT-d directly.

Figure 2.5 shows that the number of messages required by DSPH is up to 20% more than that required by ASPH during the construction of a delay constrained multicast

tree, and is up to 55% more than that required by ASPH during the on-going multicast session. This result is surprising as we know that doing fault recovery normally costs extra number of messages. Intuitively, one would expect that because DSPH is so efficient on using messages, any fault recovery approach that tries to merge the list of uncovered destinations in the failed sub-tree with the list of uncovered destinations in the active node will have a worse number of messages than DSPH even though DSPH has to run twice. Contrary to this expectation, ASPH has a significantly better message performance than DSPH. This is due to the fact that ASPH uses the approach that refrains from sending messages on node failure as explained in the 5th paragraph in section 2.2.1. The previous analysis shows that the delay on sending node failure messages does not have a negative effect on the quality of the generated multicast trees.

Figure 2.5: Number of messages versus group size when node failure occurs

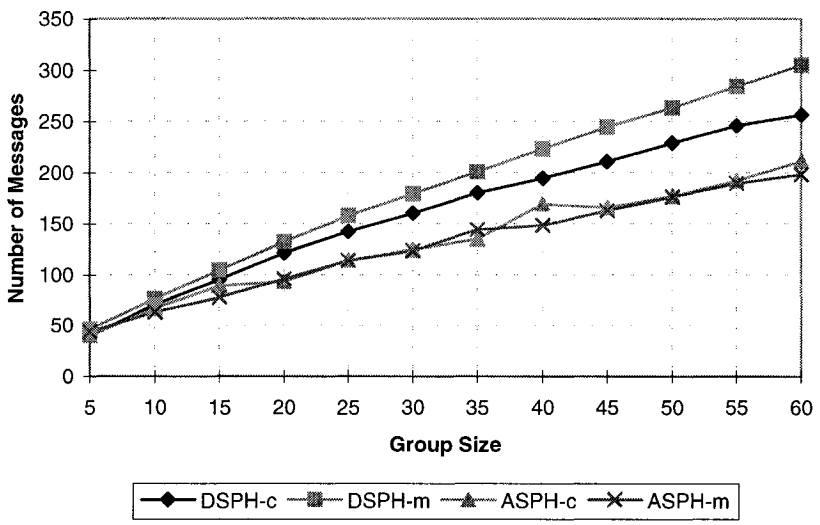
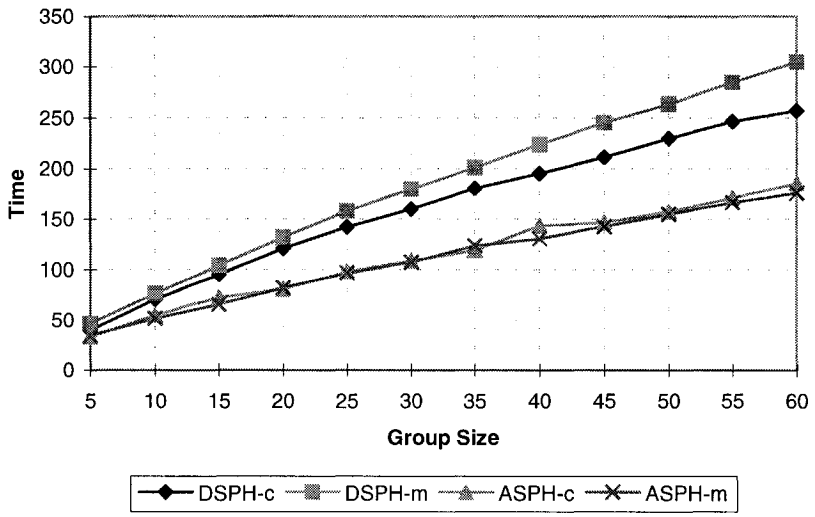


Figure 2.6 shows that the convergence time required by DSPH is up to 50% and 75% more than that by ASPH. This result confirms what has been expected. Through the

localized recovery approach taken by ASPH, it is expected that the convergence time can be reduced by ASPH.

Figure 2.6: Time versus group size when node failure occurs



When the group size is fixed at 20 in 200-node networks and Δ varies between d_{max} + $(1/8)d_{max}$ and d_{max} + $(23/8)d_{max}$, where d_{max} = max({d_u | for any $u \in S$: d_u is the delay on the shortest path from s to u}), Figure 2.7, Figure 2.8 and Figure 2.9 give the corresponding results. Figure 2.7 shows that the costs of the generated trees by two algorithms are almost identical. Also, confirming what has been expected, when the delay constraint is tight, the cost of the generated tree is slightly higher than that when the delay constraint is relaxed since the algorithm has less options to optimize the tree. Figure 2.8 and Figure 2.9 show that the message and time performance for ASPH is much better than that for DSPH. For DSPH, they show that the tighter the delay constraint, the higher the number of messages and time. This could be due to the fact that when the delay constraint is tight, DSPH tends to fork more tree paths directly from the source node. However, ASPH behaves contrary to DSPH. This may be due to the localized recovery approach

used in ASPH, which tends to try less alternatives to expand the tree under tighter delay constraints.

Figure 2.7: Tree cost versus delay constraint when node failure occurs

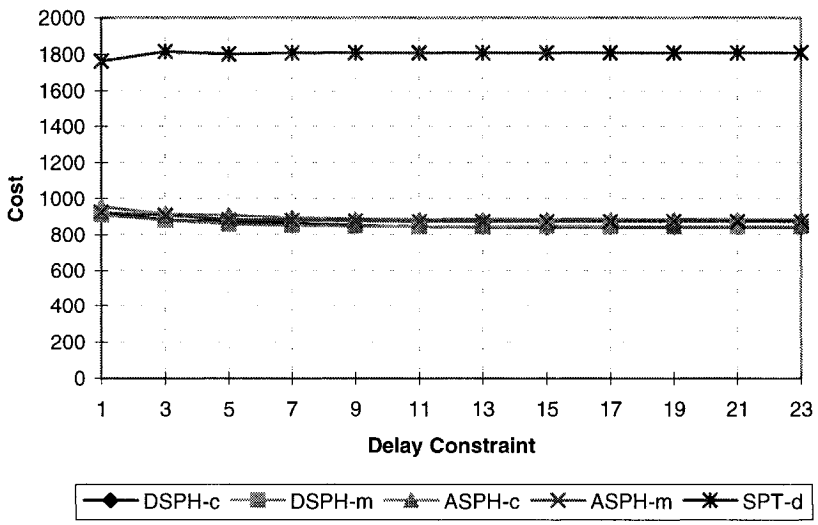


Figure 2.8: Number of messages versus delay constraint when node failure occurs

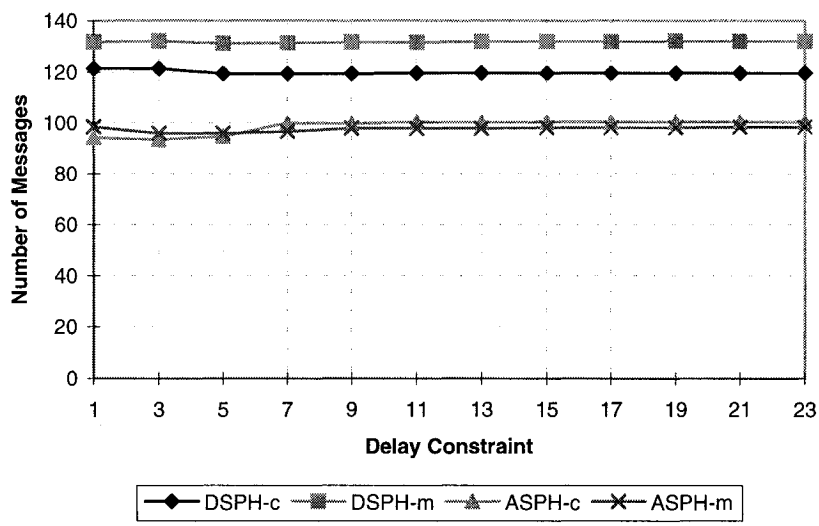
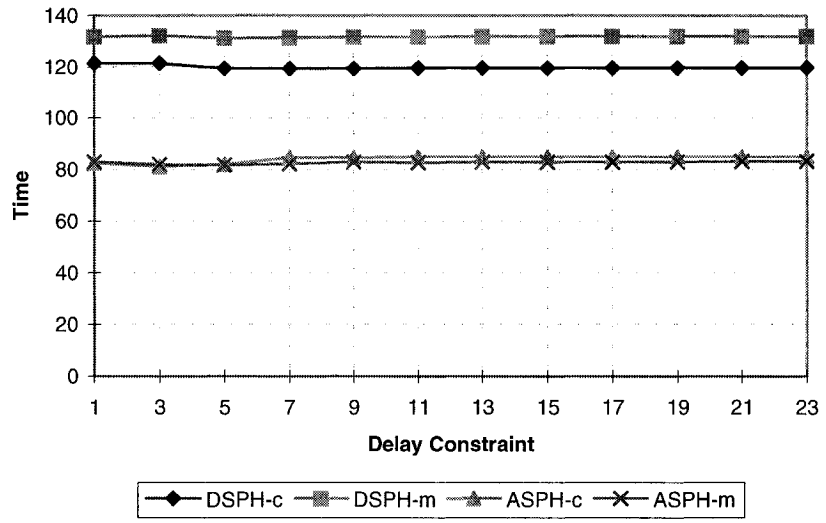


Figure 2.9: Time versus delay constraint when node failure occurs



2.3.2 Link failure

In the simulations, link failures are injected into networks in order to see how the multicast routing algorithms perform in a network where link failures occur. There are two types of experiments: one is for the case when link failures occur during the construction of a multicast tree, and the other is for the case when link failures occur during the on-going multicast session. For the first type of experiments, it is assumed that at most one link failure may occur during the construction of the multicast tree. The timing for a link failure is randomly selected so it could occur randomly among the different stages of the construction of the multicast tree. The failed link is randomly selected among the links in the multicast tree built. DSPH will be re-run when a link failure occurs.

The number of messages exchanged, the convergence time and the cost of the multicast tree are measured by their average value in a total of 100 simulation runs on a network with 200 nodes. Note that an exchanged message will only be counted once from its sender to its receiver no matter how many intermediate nodes are walked through as

long as the algorithm running on the node does not interpret the message. Meanwhile, the convergence time is counted by taking one message exchange as a time unit. However, within one time unit, there may be several message exchanges occurring in the network. Thus, multiple messages exchanged within the same time unit will only be counted once in the convergence time. The size of the messages is not taken into consideration in the performance measurement.

For the second type of experiments, DSPH will have to be re-run to re-build the entire multicast tree when a link failure occurs during an ongoing multicast session. In the experiments, one multicast tree link will be randomly selected as a failure link during a multicast session for the type of networks that we study. Like the first type of experiments, the number of messages exchanged, the convergence time and the cost of the multicast tree are measured by their average value in a total of 100 simulation runs on a network with 200 nodes.

As DSPH will be re-run when a link failure occurs while ASPH will always return a multicast tree no matter whether a link failure occurs or not, ASPH has a better success rate than DSPH.

Figure 2.10, Figure 2.11 and Figure 2.12 show the simulation results when Δ is set to $d_{max} + (3/8)d_{max}$, where $d_{max} = \max(\{d_u \mid \text{for any } u \in S\text{: } d_u \text{ is the delay on the shortest path from } s \text{ to } u\})$, and the group size changes between 5 and 60 in 200-node networks. In the figures, suffix "-c" means during the construction of a multicast tree, suffix "-m" means during the on-going multicast session and "SPT-d" means the delay-based SPT. The delay-based SPT could be considered as the delay constrained multicast trees without any optimization on tree costs.

Figure 2.10: Tree cost versus group size when link failure occurs

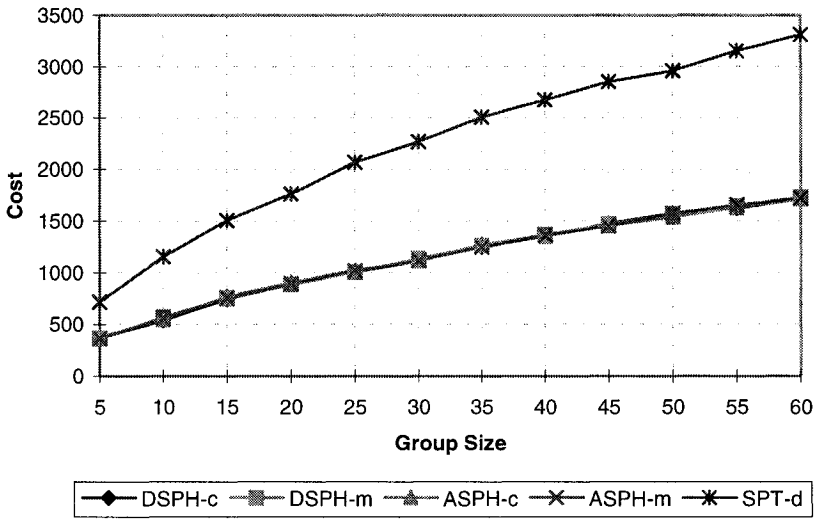


Figure 2.10 shows that the costs of the trees generated by ASPH are almost identical to those by DSPH. This result is very encouraging. DSPH calculates the multicast tree based on the consistent current network topology information after it re-runs while ASPH might use different network topology information for different parts of a delay constrained multicast tree. Intuitively, it could be expected that the costs of the trees generated by ASPH should be noticeably higher than those by DSPH. But, the simulation results show that it is not the case. It is also shown in Figure 2.10 that the delay-constrained multicast tree algorithms generate trees with much better cost performance than the algorithms without considering optimization on the tree cost such as SPT-d. This means that it is worth using the delay-constrained multicast tree algorithms rather than using SPT-d directly.

Figure 2.11 shows that the number of messages required by DSPH is up to 59% more than that required by ASPH during the construction of a delay constrained multicast

tree, and is up to 63% more than that required by ASPH during the on-going multicast session. This result is as surprising as what we found in the node failure case. Similar to the analysis done in the node failure case, this result can be attributed to the fact that ASPH uses the approach that refrains from sending messages on link failure. This shows that the delay on sending link failure messages does not have a negative effect on the quality of the generated multicast trees.

Figure 2.11: Number of messages versus group size when link failure occurs

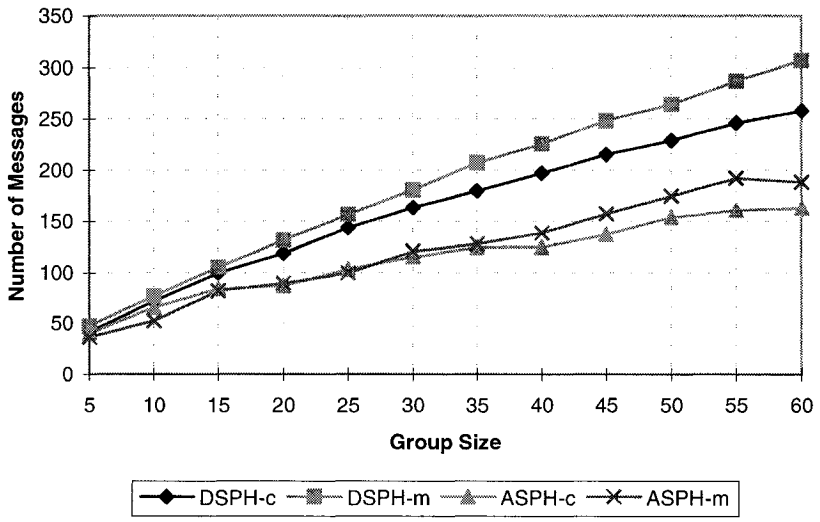


Figure 2.12: Time versus group size when link failure occurs

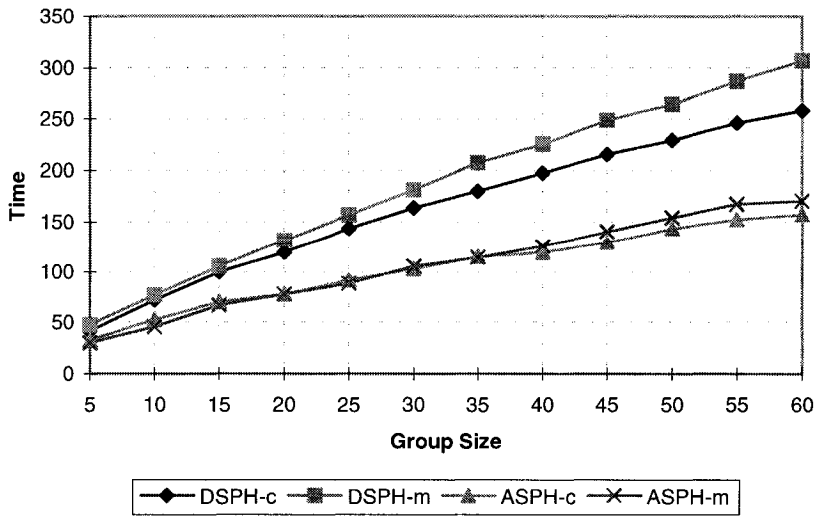
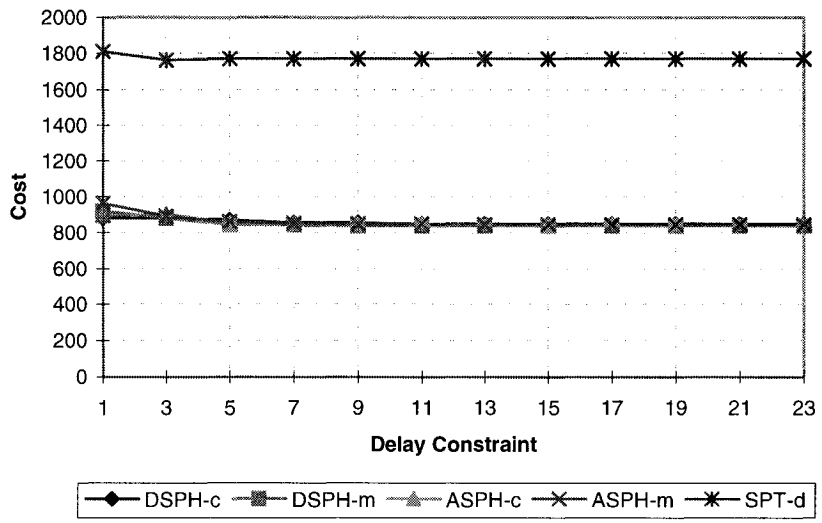


Figure 2.12 shows that the convergence time required by DSPH is up to 67% and 81% more than that by ASPH. This result confirms what has been expected. Through the localized recovery approach taken by ASPH, it is expected that the convergence time can be reduced by ASPH.

Figure 2.13: Tree cost versus delay constraint when link failure occurs



When the group size is fixed at 20 in 200-node networks and Δ varies between d_{max} + $(1/8)d_{max}$ and d_{max} + $(23/8)d_{max}$, where $d_{max} = \max(\{d_u \mid \text{for any } u \in S$: d_u is the delay on the shortest path from s to $u\})$, Figure 2.13, Figure 2.14 and Figure 2.15 give the corresponding results. Figure 2.13 shows that the costs of the generated trees by two algorithms are almost identical. Also, confirming what has been expected, when the delay constraint is tight, the cost of the generated tree is slightly higher than that when the delay constraint is relaxed since the algorithm has less options to optimize the tree. Figure 2.14 and Figure 2.15 show that the message and time performance for ASPH is much better than that for DSPH.

Figure 2.14: Number of messages versus delay constraint when link failure occurs

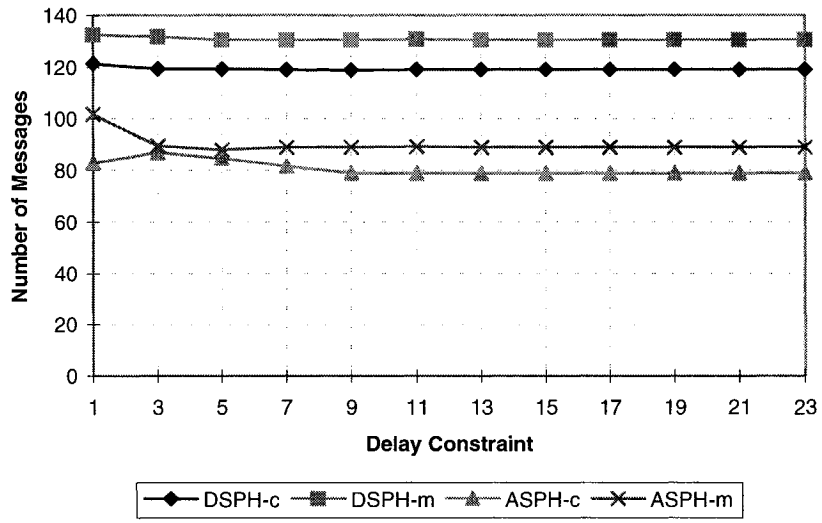
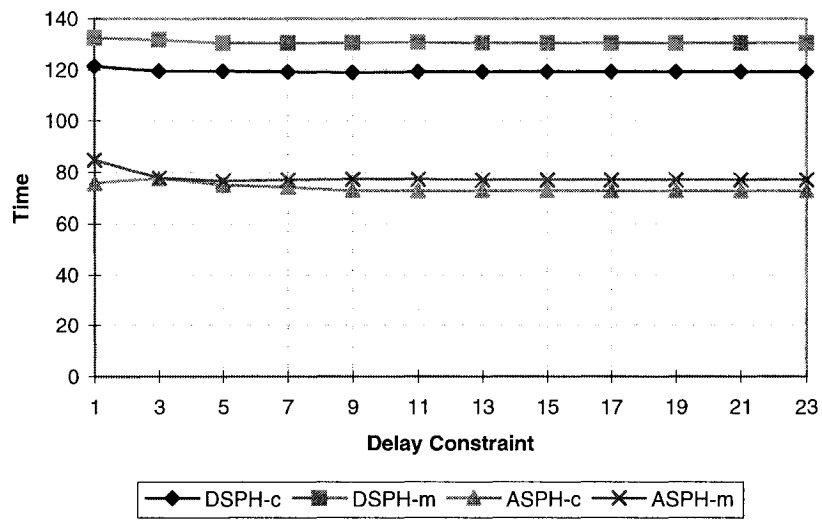


Figure 2.15: Time versus delay constraint when link failure occurs



In conclusion, ASPH performs better than DSPH in terms of the number of exchanged messages and the convergence time in addition to the advantage that ASPH has local recovery from the node or link failures without re-building the entire tree. Re-building the entire multicast tree will cause the interruption of the multicast session, which is avoided in ASPH. Meanwhile, the costs of the generated multicast trees by two

algorithms are almost identical, which means that the quality of the multicast trees generated by ASPH is as high as that generated by DSPH even though ASPH conducts the fault recovery actions.

2.4 Conclusions

In this chapter, we proposed a new distributed SP-based delay constrained multicast routing algorithm which takes into account the changes in the topology of the network. The proposed algorithm can recover from node or link failures during the construction of a delay constrained multicast tree, and during an on-going multicast session without requiring the rebuilding of the entire multicast tree. Furthermore, compared with the existing distributed SP-based delay constrained multicast routing algorithm DSPH that uses the naïve fault recovery approach, the proposed algorithm gives better performance in terms of the number of exchanged messages and the convergence time when applied in a network where node or link failures occur.

Chapter 3

Fault Recovery for a Distributed QoS-based Multicast Routing Algorithm

3.1 Introduction

In this chapter, we propose a new MST based distributed multicast routing algorithm which will give a sub-optimal solution to a constrained multicast tree problem taking into consideration the node failures in a network. The analysis and simulation results show that the proposed algorithm gives better performance not only in terms of the success rate, but also in terms of the number of exchanged messages and the convergence time in a network where node failures occur.

The rest of this chapter is organized as follows. Section 3.2 describes the new algorithm; Section 3.3 discusses the performance of the new algorithm; and finally Section 3.4 gives the conclusions.

3.2 Distributed multicast routing using MST approach

3.2.1 Overview of the proposed MST-based algorithm

The algorithm proposed in this chapter, as an MST based delay constrained multicast routing algorithm, is similar to the *Distributed KPP heuristic algorithm* (DKPP) described in [KPP96]. The difference is that the proposed algorithm is able to construct a constrained multicast tree even when a node failure occurs in the network during the tree

construction period, and to recover from a node failure in a constrained multicast tree during an on-going multicast session without interrupting the traffic on the unaffected portion of the tree. The proposed algorithm, like DKPP, is based on a distributed minimum spanning tree algorithm such as the one described in [GHS83], which essentially mimics Prim's MST algorithm [Prim57][CLR92]. In the proposed algorithm, as well as in DKPP, several modifications have been made on a distributed MST algorithm to ensure that the algorithm will generate a multicast tree in a network that meets the delay bound:

1. instead of allowing the tree to be built up in fragments that eventually connect together, the constrained multicast tree is built from a single root. This corresponds to the case where the distributed MST algorithm is invoked by a single node.
2. instead of letting the points of decision, on which edge should be added next, move to the two endpoints of a newly added edge in the tree, the point of decision is always at the source.
3. instead of ignoring the restrictions on the delay when a tree is expanded, the new selection function will be "selecting the lowest cost edge out-going from the sub-tree constructed so far such that the delay constraint is not violated and no cycle is introduced".
4. a cycle make-and-break technique is used to solve the problem that some destination nodes cannot be reached within the delay constraint from the tree built so far. After adjusting the structure of the tree built so far, those destination nodes can be reached.

The proposed algorithm will adaptively construct a delay constrained multicast tree even when the network topology changes (e.g., when node failures occur). Thus, the

proposed algorithm is called *Adaptive Distributed Minimum spanning tree based Heuristic* (ADMH).

Informally, given a network $G=(V, E)$, a source node s ($s \in V$), a set of destination nodes S ($S \subseteq V - \{s\}$), and a delay constraint Δ, ADMH progresses as follows:

1. starts with a tree containing only the source node s;
2. selects a node v in G which is not in the tree yet (called *non-tree node*) and can be reached by the least cost edge from the tree under the delay constraint and without introducing a cycle. Then the edge from the tree to v is added to the tree;
3. checks if any tree node fails. When a failure is detected, removes the sub-tree rooted at the failure node and returns the destination nodes covered by the sub-tree back to the list of destinations which are not yet covered by the tree.
4. repeats steps 2-3 until all nodes in S are included in the tree. Prune the tree of unnecessary edges.

Step 3 is the step introduced in ADMH to deal with the node failures occurring during the construction of the tree.

After the tree is constructed and during an ongoing multicast session, ADMH continues to run the following steps to repair the multicast tree if node failures occur:

5. repeats step 3 until a node failure is detected.
6. repeats steps 2-3 until all nodes in the new S are included in the tree. Prune the tree of unnecessary edges.
7. repeats steps 5-6 until the multicast session is closed.

3.2.2 Details of the proposed MST-based algorithm

If no node failure occurs in the network, ADMH works as DKPP. The constrained multicast tree is built from a single root - source node. Each round of selection selects a best out-going edge of the tree towards a destination node that is not yet covered and adds the node reached by the selected edge into the tree. So, the tree is expanded edge by edge (or node by node) until all destination nodes are included in the tree. To facilitate the selection of a best out-going edge (which has the least cost edge from the tree under the delay constraint and whose inclusion into the tree will not form a cycle), each out-going edge of a tree node will be classified as one of the following during each round of selection:

- *Potentially Usable* - A *Potentially Usable* edge is an edge that can potentially be selected because there is at least one uncovered destination that can be reached through it without exceeding the delay constraint.
- *Unusable* - An *Unusable* edge is an edge that is not *Potentially Usable*.
- *Cycle* - A *Cycle* edge is an edge that is *Potentially Usable*, but its inclusion to the tree would create a cycle.
- *Usable* - A *Usable* edge is an edge that is *Potentially Usable*, but not a *Cycle* edge.

The following expression is used to determine whether an edge is *Potentially Usable* or *Unusable*; that is, to determine the reachability of destination nodes through a neighbor node w of a tree node v:

$$\exists d \in S', P(v) + D(v,w) + SD(w, d) < \Delta$$

where $SD(w,d)$ = minimum delay from w to d; $D(v,w)$ = delay on the link (v,w); $P(v)$ = delay on path from s to v in the tree; S' is the set of uncovered destinations. If the

expression is true, edge (v, w) will be marked as *Potentially Usable*. Otherwise, edge (v, w) will be marked as *Unusable*. If node w of a *Potentially Usable* edge (v, w) is in the multicast tree, then edge (v, w) will be marked as *Cycle* edge. Otherwise, it is marked as *Usable* edge.

The objective of the selection function that will be used in the proposed algorithm is "selecting the lowest cost edge out-going from the sub-tree constructed so far such that the delay constraint is not violated and no cycle is introduced". In order to achieve this objective, the selection function, denoted by f_C, is applied to all *Usable* edges to help selecting the best candidate as f_C returns the cost value for each edge (v, w). The returned cost values are then compared to find the least value edge as the best candidate.

Formally, the selection function is defined as

$f_C = C(v,w)$ if $P(v) + D(v,w) < \Delta$

$f_C = \infty$ otherwise

where $C(v,w)$ = cost of the link (v,w); $D(v,w)$ = delay on the link (v,w); $P(v)$ = delay on path from s to v in the tree.

In order to determine the reachability of destination nodes, it is assumed that each node knows the minimum delay to every other node in the network. This assumption can be satisfied through running a distributed shortest path algorithm.

In some cases, it is possible that all out-going edges are marked *Unusable* due to the fact that some destination nodes cannot be reached within the delay bound from the tree built so far. In those cases, the tree cannot be expanded any further. The solution is to use a backtracking technique called "cycle make and break phase" [KPP96]. It selects a

best edge according to the selection function among *Cycle* edges, expands the tree along the *Cycle* edge and then breaks the cycle by removing one of the edges in the cycle.

When a node failure occurs, the existing MST based heuristics such as DKPP have to be re-run from scratch to re-build the entire multicast tree. ADMH recovers from the node failures such that only the part of tree affected by the failures is re-built. There are several issues that must be addressed in order to recover from a node failure. The first issue is detecting a node failure. We assume that node failures are detected by a lower level protocol which could be a very simple ping protocol. The second issue is identifying the nodes that are affected by the node failure. These nodes are the nodes that are in the sub-tree rooted at the failed node except the root of the sub-tree (i.e., the failed node). In order to start the recovery, the affected nodes must be removed from the multicast tree (constructed so far) and those destinations covered by the sub-tree are put back in the set of destination nodes that need to be attached to the final multicast tree. The third issue is determining which nodes need to know the node failure to participate in the necessary recovery. The goal here is to reduce the number of messages and the convergence time required for recovery. We found that only the source node, the nodes in the sub-tree rooted at the failed node, and the neighbor nodes of the sub-tree rooted at the failed node need to know that a node failure has occurred. The source node is responsible for adding the destination nodes in the removed sub-tree back to the set of uncovered destination nodes. The nodes in the sub-tree rooted at the failed node need to be notified the failure of the node so that they can be removed from the tree. The neighbor nodes of the sub-tree rooted at the failed node need to know the node failure so that if any of those nodes has a *Cycle* edge terminating at a node in the sub-tree rooted at the failed node, the status of the

Cycle edge is changed to *Unknown*. This will ensure that the edge is available for the competition of the best out-going edge in the next round of selection.

Following is the portion of ADMH constrained multicast routing algorithm for handling a node failure condition:

Message List:

remove - removing a sub-tree from the tree;

remove_adjust - notifying to adjust the edge status after a node is removed from the tree;

destination - adding destination back to the uncovered destination list.

1. When a node detects that one of its neighbors fails, it removes the failed node from its adjacent edge list.
2. When a tree node detects that one of its children fails, it removes the child node from the tree.
3. When a tree node detects that its parent node fails, it removes itself from the tree and sends *remove* messages to all of its children so that the sub-tree rooted at the failed node will be removed from the tree. Also, it sends *remove_adjust* message to all of its neighbors except its children. If the node is a destination node, it sends a *destination* message to the source node *s* so that the source node *s* will add the destination node back to the uncovered destination list.
4. When the tree node receives the *remove* message, the node removes itself from the tree. If the node is not a leaf node, it will forward the *remove* message to all of its children. Also, it sends *remove_adjust* message to all of its neighbors except its parent and children if it has any children. If the node is a destination node, it sends a

destination message to the source node *s* so that the source node *s* will add the destination node back to the uncovered destination list.

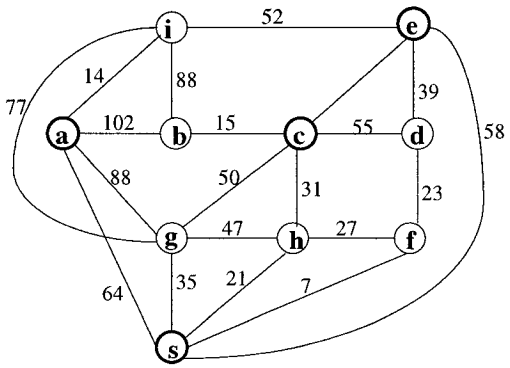
5. When a node which is a neighbor of a tree node receives the *remove_adjust* message, it will change the status of the edge, through which it received the *remove_adjust* message, to *Unknown* if the status of that edge is *Cycle*. This update of edge status is required to make the edge available as a candidate for the next round of selection of the best out-going edge.
6. When the source node *s* receives the *destination* message, it adds the sender node to the uncovered destination list.

The pseudocode of the portion of ADMH for error recovery is shown in Figure 3.1.

Figure 3.1: Pseudo-code of ADMH Algorithm (the portion for node failure recovery)

```
/* local = local node                          */
variables:
/* s = source node                             */
/* D = accumulated delay from the source       */
/* dest_list = uncovered destination list      */

initialize:
D := 0;

/* Transaction 1: */
on detecting the failure of a neighbor node
  remove the neighbor node from the adjacent
  edge list;

/* Transaction 2: */
on detecting the failure of a child node
  remove the child node from the tree;

/* Transaction 3: */
on detecting the failure of the parent node
  remove local from the tree;
  for each neighbor node do
    send remove_adjust() to neighbor;
  end
  if local is not a leaf node then
    for each child in the tree do
      send remove() to child;
    end
  else if local is a destination node then
    send destination() to s;
  endif

/* Transaction 4: */
on receiving remove()
  remove local from the tree
  D := 0
  for each neighbor node do
    send remove_adjust() to neighbor;
  end

  if local is not a leaf node then
    for each child in the tree do
      send remove() to child;
    end
  else if local is a destination node then
    send destination() to s;
  endif

/* Transaction 5: */
on receiving remove_adjust()
  if status for edge (local, sender) = Cycle then
    reset the status to Unknown;
  end

/* Transaction 6: */
on receiving destination()
  add sender to dest_list
```

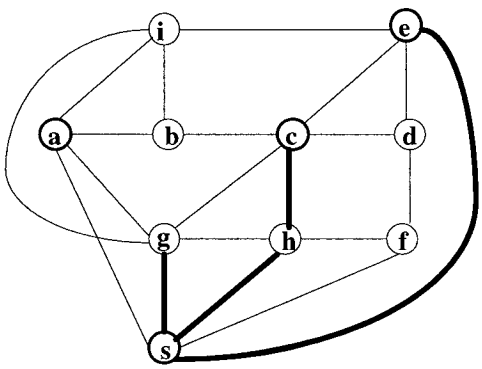
Figure 3.2 An example for ADMH Heuristic



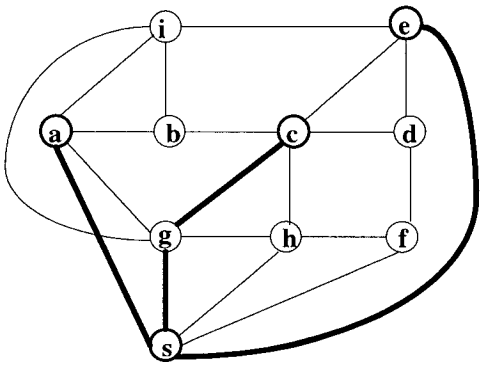
(a) a network

	s	a	b	c	d	e	f	g	h	i
s	-	10	-	-	-	36	56	11	32	-
a	17	-	3	-	-	-	-	1	-	7
b	-	48	-	44	-	-	-	-	-	9
c	-		30	-	12	2	-	24	22	-
d	-	-	-	35	-	59	50	-	-	-
e	29	-	-	3	15	-	-	-	-	38
f	6	-	-	-	55	-	-	-	22	-
g	47	48	-	30	-	-	-	-	32	22
h	34	-	-	12	-	-	55	34	-	-
i	-	30	27	-	-	37	-	40	-	-

(b) Delay *D(e)*



(c) a partial multicast tree when node *h* fails



(d) a multicast tree for (a)

Figure 3.2 shows an example which illustrates how ADMH recovers from a node failure during the construction of a constrained multicast tree. In the example, the source node is *s*, and the destination nodes are *a*, *c* and *e*. The delay values on each link are listed in the delay table in Figure 3.2(b). The delay constraint Δ is 58. For simplicity, costs on links are assumed to be equal on both directions and are shown on the graph edges in Figure 3.2(a). The multicast tree starts with source node *s* only. Then node *s* tries to find which edge is the best out-going edge. Though the cost on link (*s*, *f*) is the minimum

among all links connected with the tree but its delay 56 is so high that there is no feasible path reaching any destinations via *f*. So, link (*s*, *h*) is selected as it has the second best cost among all links connected with the tree and it can reach destinations *c* and *e* within Δ. Similarly, links (*h*, *c*), (*s*, *g*) and (*s*, *e*) will be added to the tree in order. At this stage (as shown in Figure 3.2(c)), two destination nodes *c* and *e* have been covered and link (*g*, *c*) is marked as *Cycle*. However, just before the algorithm tries to cover the last destination *a*, node *h* fails. As a result, the destination *c* has to be put back in the uncovered destination list and link (*g*, *c*) is marked as *Unknown*. Then, links (*g*, *c*) and (*s*, *a*) are selected in order. The final multicast tree is shown in bold in Figure 3.2(d).

3.2.3 Analysis of the proposed algorithm

One of the questions to be answered is if ADMH generates a constrained multicast tree. The answer is affirmative, and it is proved by the following theorem.

Theorem 3.1: When ADMH successfully constructs a multicast tree, the multicast tree is constrained; that is, the tree satisfies the delay constraint.

Proof: If there is no node failure, ADMH behaves the same as DKPP. As DKPP is claimed to generate constrained multicast trees, ADMH also generates constrained multicast trees. According to both algorithms, in order to make the final multicast tree satisfy the delay constraint, the partial multicast tree created at each step must satisfy the delay constraint as well.

When a node failure occurs, the sub-tree rooted at the failed node is removed from the multicast tree in ADMH. As stated above, we know that the tree built so far satisfies the delay constraint just before the node failure occurs. Thus, the remaining tree rooted at the source still satisfies the delay constraint since the node failure only affects the delay

from the source to any node in the removed sub-tree. ADMH continues to expand the tree while ensuring that the delay constraint is satisfied in the same way as it does before the node failure occurs. Therefore, the multicast tree constructed by ADMH satisfies the delay constraint. EOP.

The remaining questions to be answered are about the performance of ADMH. Will ADMH indeed have a better performance than DKPP in terms of the number of message exchanges and the convergence time? What are the quality of the generated multicast trees by ADMH in terms of the tree cost? Though the following theorem shows that in the worst case, ADMH performs at least as good as DKPP in terms of the number of messages exchanged, the simulation conducted in the next section does indicate that ADMH performs much better than DKPP on average case in terms of both the number of message exchanges and the convergence time. The simulation also shows that the multicast trees generated by ADMH are as good as those by DKPP in terms of the tree cost.

Theorem 3.2: The number of messages exchanged by algorithm ADMH is $O(n^3)$ in the worst case.

Proof: As shown in [KPP96], the number of exchanged messages for DKPP is $O(n^3)$. When there are no node failures in the network, ADMH behaves the same as DKPP. So, the number of exchanged messages for ADMH in the worst case is at least $O(n^3)$. The extra messages exchanged in ADMH are the messages for the recovery from node failures. $O(n)$ *remove* messages may be exchanged to remove the sub-tree rooted at the failed node. $O(n)$ *destination* messages may be sent to add the destination nodes back to the uncovered destination list. Finally, $O(n^2)$ *remove_adjust* messages may be sent to the

neighbors of the removed sub-tree. So, the number of messages exchanged by ADMH is $O(n^3)$ in the worst case. EOP.

3.3 Performance analysis

In order to compare the performance of ADMH with DKPP, a series of simulations have been performed by applying both DKPP and ADMH to networks generated by the Waxman's approach, as explained in Section 1.4. For simplicity, the cost of a link is made equal to its delay in the simulations, in this case.

3.3.1 Experiments during the construction of a multicast tree

In the simulations, node failures are injected into networks in order to see how the multicast routing algorithms perform in a network where node failures occur. It is assumed that at most one node failure may occur during the construction of the multicast tree. The timing for a node failure is randomly selected so it could occur randomly among the different stages of the construction of the multicast tree. The failed node is randomly selected among the nodes that are neither the source node nor the destination nodes when node failure occurs, since the failure of the source node means that there will be no multicast tree to be built and the failure of a destination node means that the constructed multicast tree will not be comparable with other multicast trees which cover all destination nodes.

As DKPP will be re-run when a node failure occurs while ADMH will always return a multicast tree no matter whether a node failure occurs or not, ADMH has a better success rate than DKPP.

The number of messages exchanged, the convergence time and the cost of the multicast tree are measured by their average value in a total of 100 simulation runs on a

network with 200 nodes. Δ is set to $d_{max} + (1/2)d_{max}$, where $d_{max} = \max(\{d_u \mid \text{for any } u \in S$: d_u is the delay on the shortest path from s to $u\}$). Note that an exchanged message will only be counted once from its sender to its receiver no matter how many intermediate nodes are walked through as long as the algorithm running on the node does not interpret the message. Meanwhile, the convergence time is counted by taking one message exchange as a time unit. However, within one time unit, there may be several message exchanges occurring in the network. Thus, multiple messages exchanged within the same time unit will only be counted once in the convergence time.

Figure 3.3, Figure 3.4 and Figure 3.5 show the simulation results when the failure occurs during the construction of a multicast tree. It can be seen that the performance on the number of messages and time for ADMH is much better than that for DKPP under the condition that a node may fail during the construction of a multicast tree. DKPP requires 45% to 60% more number of messages and 55% to 70% more time than ADMH. On the other hand the costs of the generated trees by two algorithms are almost identical.

Figure 3.3: Messages versus group size when a tree node failure occurs during the construction of a multicast tree

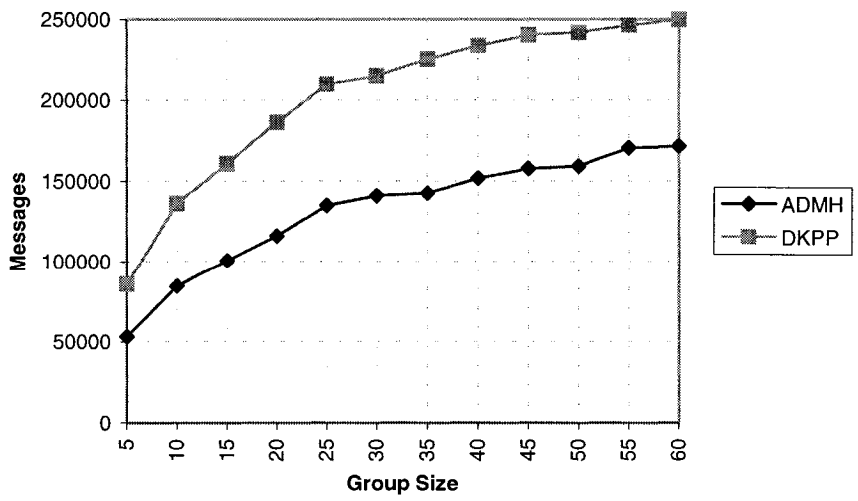


Figure 3.4: Time versus group size when a node failure occurs during the construction of a multicast tree

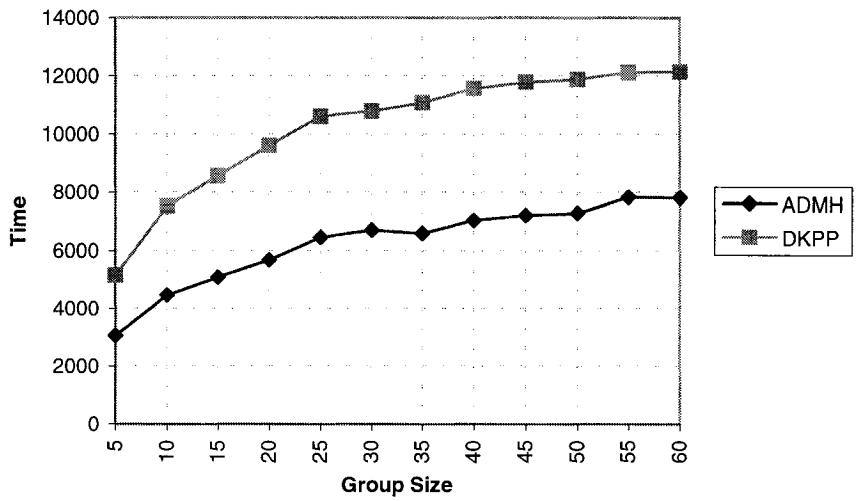
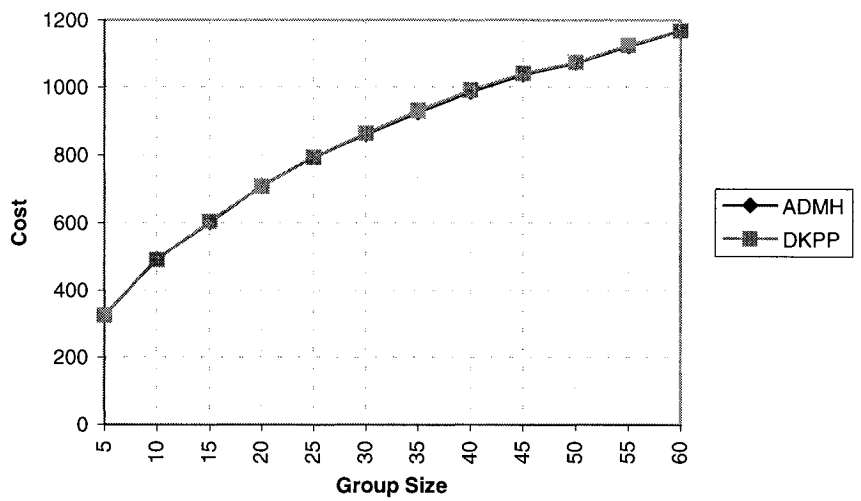


Figure 3.5: Tree cost versus group size when a node failure occurs during the construction of a multicast tree



3.3.2 Experiments after the construction of a multicast tree

DKPP will have to be re-run to re-build the entire multicast tree when a node failure occurs during an ongoing multicast session. In the experiments, one multicast tree node

will be randomly selected as a failure node during a multicast session for the type of networks that we study. Like the experiments done in the previous section, the number of messages exchanged, the convergence time and the cost of the multicast tree are measured by their average value in a total of 100 simulation runs on a network with 200 nodes.

From the simulation results shown in Figures 3.6 and 3.7, ADMH performs 70% to 90% better than DKPP in terms of the number of exchanged messages and the convergence time in addition to the advantage that ADMH has local recovery from the node failure without re-building the entire tree. Re-building the entire multicast tree will cause the interruption of the multicast session, which is avoided by the proposed algorithm.

Figure 3.6: Messages versus group size when a node failure occurs during a multicast session

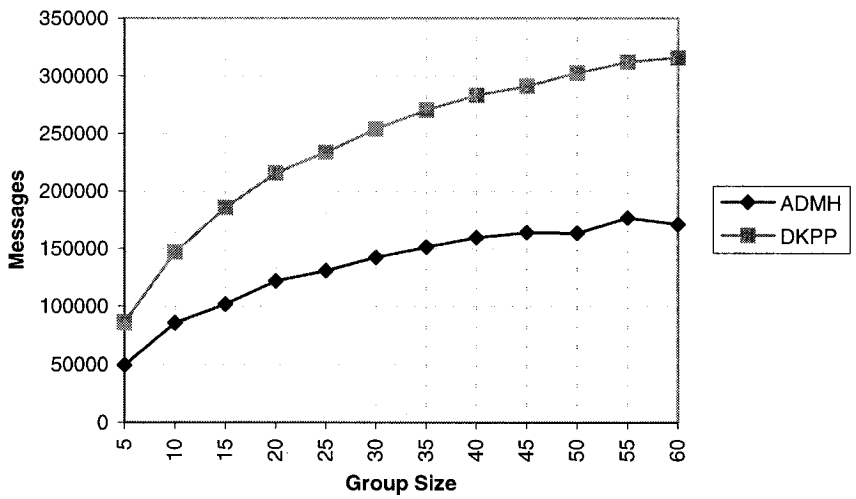


Figure 3.7: Time versus group size when a node failure occurs during a multicast session

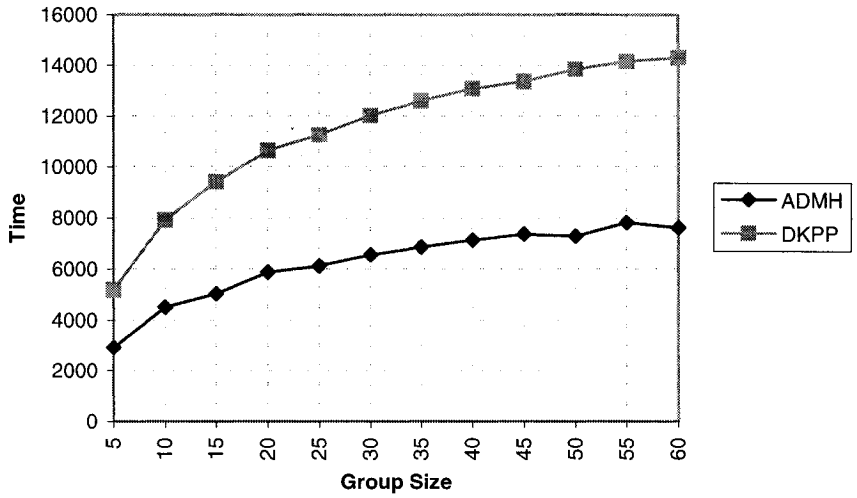


Figure 3.8: Tree cost versus group size when a node failure occurs during a multicast session

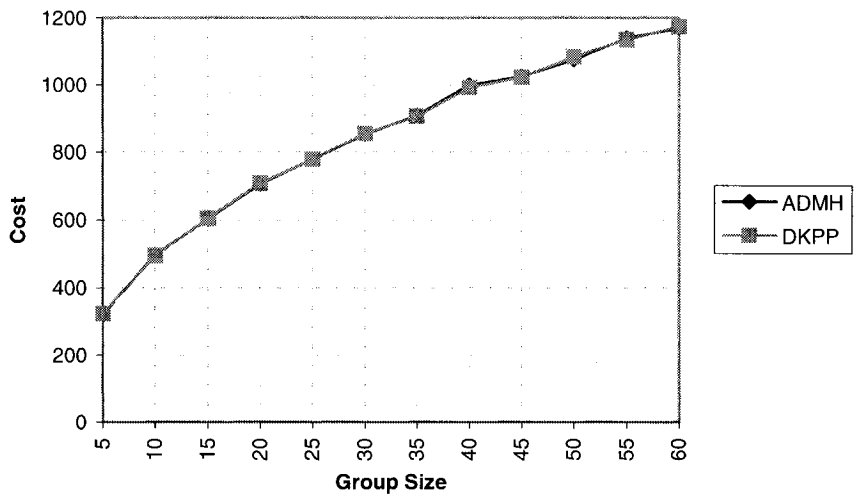


Figure 3.8 shows that the costs of the generated multicast trees by two algorithms are almost identical, which means that the quality of the multicast trees generated by ADMH is as high as that generated by DKPP even though ADMH conducts the fault recovery actions.

3.4 Conclusions

In this chapter, we proposed a new MST based distributed delay-constrained multicast routing algorithm which takes into account the changes in the topology of the network. Compared with the existing MST based distributed delay-constrained multicast routing algorithms such as DKPP, the proposed algorithm gives better performance in terms of the success rate, the number of exchanged messages and the convergence time when applied in a network where node failures occur. Furthermore, the proposed algorithm can recover from node failures during an on-going multicast session without requiring the rebuilding of the entire multicast tree and interrupting the service to the unaffected portion of the multicast tree.

Chapter 4

An Efficient Distributed QoS-based Multicast Routing Algorithm

4.1 Introduction

An MST based algorithm like DKPP runs with a high complexity of $O(n^3)$ in the number of messages and time. Secondly, an MST based algorithm has a very low success rate (which is the number of trials to build a multicast tree successfully divided by the number of total trials) in constructing a constrained multicast tree especially under tight delay constraints. On the other hand, an SP based algorithm like DSPH has a fatal deficiency; that is, the algorithm will not be able to find a solution if the delay constraint is less than the maximum delay of a cost based shortest path tree as DSPH only considers cost based shortest paths. It is obvious that a delay constrained multicast tree exists as long as the delay constraint is larger than the maximum delay of a delay based shortest path tree. This means that DSPH may not solve the delay constrained multicast tree problem for a large number of cases where a delay constraint multicast tree exists. Lastly, both algorithms try to construct the multicast tree sequentially without taking advantage of the concurrency in a distributed algorithm. Thus, they take a long time to construct a multicast tree.

In this chapter, we propose an efficient distributed multicast routing algorithm which builds the multicast tree in a concurrent manner and thus runs with a very low message and time complexity of $O(mn)$ and $O(n)$ respectively. Also, the proposed

algorithm has a near zero failure rate in constructing a multicast tree even under tight delay constraints without any limitation on the value of delay constraints.

The rest of this chapter is organized as follows. Section 4.2 describes the proposed algorithm; Section 4.3 discusses the performance of the proposed algorithm; and finally Section 4.4 gives the conclusions.

4.2 A concurrent distributed multicast routing algorithm

4.2.1 Overview of the proposed concurrent algorithm

The algorithm proposed in this chapter is an SP based distributed multicast routing algorithm. The reason for proposing an SP based algorithm rather than an MST based algorithm is that our goal is to find a distributed multicast routing algorithm with low message and time complexity, but the MST based distributed multicast routing algorithms use a distributed MST algorithm which inherently requires high message and time complexity. Before we present the proposed algorithm, we define the related terminology. A *cost based shortest path* from node v to destination d is a path from v to d which has the minimum cost. A *delay based shortest path* from node v to destination d is a path from v to d which has the minimum delay. A *delay (cost) based shortest path tree* rooted at node s to the set S of destination nodes is a tree consisting of delay (cost) based shortest paths from s to each destination $d \in S$. A *feasible* cost (delay) based shortest path from node v to destination d is a cost (delay) based shortest path from v to d which satisfies the delay constraint. A *potentially feasible* cost based shortest path from node v to destination d is a cost based shortest path from v to d where the immediate successor w of v on the path can reach d under the delay constraint; that is, $P(v) + D(v,w) + SD(w,d) < \Delta$, where $SD(w,d) =$

minimum delay from w to d; $D(v,w)$ = delay on the link (v,w); $P(v)$ = delay on path from s to v in the tree.

The proposed algorithm takes the following approach to overcome the shortcomings that exist in DKPP and DSPH while maintaining the quality of the multicast tree (i.e., tree cost) as optimal as possible:

1. DKPP only assumes that the link cost to each neighbor is known to the local node and expands the multicast tree edge by edge which causes high number of messages and time complexity. Based on the fact that the existing routing information is available at each node, we will use this information to derive the information about the cost based shortest path from the local node to each node in the network and make the derived information available to the local node so that the SP algorithm can be applied. As in DKPP, it is also assumed that the information about the delay based shortest path from the local node to each node in the network is known to the local node.

   DSPH is an SP based algorithm. However, the multicast tree is expanded sequentially to cover each destination. This means that it could take a long time for the algorithm to complete the construction of a multicast tree. The proposed algorithm will expand the multicast tree to cover all destinations in a concurrent manner. The criterion used to expand the multicast tree will be "expand the tree from the local node along the cost based shortest path if the delay constraint is satisfied; otherwise expand the tree along the delay based shortest path". This is different from DSPH 's criterion which is "expand the tree along the cost based shortest path from the tree to a destination node" that can only be applied sequentially. Also, the criterion that we will use allows the proposed algorithm to include a cost based shortest path to a destination as much as

possible in order to make the tree cost as low as possible. It is important to note that conflicts may occur if a node is invited more than once to join the tree. The solution is that the node where the conflicts occur will choose its parent whose accumulated delay from the source node is minimal to ensure that all destinations that go through this node will satisfy the delay constraint.

2. DSPH requires that the delay constraint is larger than the maximum delay of all cost based shortest paths from the source to each destination. This means that DSPH will not be able to find a delay constrained multicast tree when the delay constraint is smaller than the maximum delay of the cost based shortest paths from the source to each destination, but larger than the maximum delay of the delay based shortest paths from the source to each destination (i.e., in the cases that there should exist delay constrained multicast trees). The proposed algorithm will not have such a limitation due to the fact that the tree expansion criterion described in 1 above is used.

The proposed algorithm is called *Distributed Concurrent Shortest Path heuristic* (DCSP) since it tries to cover all destinations concurrently. The concurrency in this context means how the destinations are to be covered, in contrast to the sequential approach used in the existing distributed delay-constrained multicast routing algorithms where destinations are covered sequentially one by one.

Informally, given a network $G=(V, E)$, a source node s, a set of destination nodes S, and a delay constraint Δ, DCSP progresses in one or two phases as follows:

Phase I - Setup

1. Starts with a tree containing only the source node s;

2. Selects one of its neighbors as a "best" neighbor for each destination node to be covered, and expands the tree concurrently along these best neighbors. The "best" neighbor for a destination node will be the neighbor along which there is a potentially feasible cost based shortest path that can reach the destination;
3. If there is no potentially feasible cost based shortest path found, adds the destination node under consideration to the list N of destinations which are not yet covered by the tree;
4. If a node is included into the tree by more than one parent node, the node will choose the parent node which has the smallest accumulated delay from the source node;
5. Repeats steps 2-4 until each node in S is either included in the tree or included in the list N of destinations which are not yet covered by the tree.

Phase II - Adjustment

1. Starts from the source node with a tree containing all nodes S - N that are included in Phase I;
2. Selects a "best" neighbor for each destination node to be covered, and expands the tree concurrently along these best neighbors. The "best" neighbor for a destination node in this stage will be the neighbor along which there is a feasible delay based shortest path;
3. If there is no feasible delay based shortest path found, adds the destination node under consideration to the list M of destinations which are not yet covered by the tree;
4. If a node is included into the tree by more than one parent node, the node will choose the parent node which has the smallest accumulated delay from the source node;
5. Repeats steps 2-4 until each node in N is included in the tree or included in the list M of destinations which are not yet covered by the tree.

Phase II will be invoked only if Phase I cannot cover all destination nodes in *S*. Phase II terminates with two possible outcomes depending on whether the list *M* of uncovered destinations is empty or not. If *M* is empty, the algorithm returns a delay-constrained multicast tree. Otherwise, the algorithm returns a failure. One could use the given delay information to build a delay based shortest path tree which can be considered as a non-optimal delay-constrained multicast tree. It can be seen that 2 phases are required in the proposed algorithm as no backtracking mechanism is used in case there is no potentially feasible cost based shortest path found during the setup of a delay-constrained multicast tree.

4.2.2 Details of the proposed concurrent algorithm

Seven types of messages are used in DCSP, which are

open - opening a multicast connection;

setup - setting up a delay constrained shortest path from the source to a destination node;

break - notifying its parent to break a loop;

reject - rejecting the invitation to join the tree as either the constraints may be violated or loops may be formed;

destination - adding destination to the uncovered destination list;

notify - notifying the source node that a destination node has been covered;

adjust - adjusting the tree along the delay shortest path from the source to a destination node.

The pseudocode of DCSP is shown in Figure 4.1.

Figure 4.1: Pseudo-code of DCSP Algorithm

```
/* local = local node                                    */
                                        variables:
                                        /* s = source node                          */
                                        /* D = accumulated delay from the source    */
```

```
/* S' = destinations under consideration      */
/* destTable = node to destination table      */
/*    via - selected neighbor node            */
/*          for reaching the destination      */
/*    state - local state for the destination */
/* RouteC = route table by cost               */
/* RouteD = route table by delay              */
/*    Both RouteC and RouteD have             */
/*    next - next hop node                    */
/*    cost - the distance to a destination    */
/*    delay - the delay to a destination      */
/* destS = set of destination nodes           */
/* US = list of destinations to be covered    */
/* CS = list of destinations covered          */

initialize:
D := 0;
set via and state in destTable as unknown;
```

PHASE I: Setup

```
/* Transaction 1: */
on receiving open(S, Δ):
  s := local;
  add s to the tree;
  D := 0;
  S' := S;

  /* Transaction 1.1: select neighbors for
     expansion */
  for each d_i ∈ S' and
    destTable(d_i).state = unknown do
    destTable(d_i).via := neighbor n_i via which
        there is a potentially feasible cost based
        shortest path reaching d_i;
  end

  /* Transaction 1.2: expand the tree */
  for each n_j: ∃ d_i ∈ S', n_j = destTable(d_i).via
  do
    destS := {d_i | n_j = destTable(d_i).via, d_i ∈ S'}
    D' = D + delay(local, n_j);
    send setup(destS, D', s, S, Δ) to n_j;
    destTable(d_i).state := setup;
  end

/* Transaction 2: */
on receiving setup(destS, D, s, S, Δ)
  destS := setup.destS;
  /* Transaction 2.1: loop checking */
  if node local is already in the tree then
    /* 2.1.1: reject the setup request */
    if ∀dest∈destS, D + RouteD(dest).delay < Δ
    then
      send reject() to sender;
      S' := S' ∪ destS;
    /* 2.1.2: accept the new setup request and
      break from the old tree */
    else if setup.D < D then
      send break() to parent;
      add node local to the tree;
      D := setup.D;
      S' := S' ∪ destS;
    /* 2.1.3: the delay constraint may be
      violated and deny the setup request */
    else
      send deny(destS) to sender;
    endif
  else /* It is OK to add node local to the tree */
  /* Transaction 2.2: destination reached */
    add node local to the tree;
    D := setup.D;
    if ∃dest ∈ destS, local = dest then
      send notify(dest) to s;
      S' := S' - {dest};
    endif
  endif

  /* Transaction 2.3: select neighbors for
       expansion and expand the tree. */
    execute Transaction 1.1 and 1.2;
    /* no neighbor found */
    for each d_i∈ S':destTable(d_i).via= unknown
    do
      destS := { d_i }
      send destination(destS) to s;
      destTable(d_i).state := infeasible;
    end

/* Transaction 3: */
on receiving destination(destS)
  add destS to US
  if CS ∪ US = S then
    if US = ∅ then stop;
    else enter phase II;
    endif
  endif



/* Transaction 4: */
on receiving notify(dest)
  add dest to CS
  if CS ∪ US = S then
    if US = ∅ then stop;
```

```
    else enter phase II;
    endif
  endif

/* Transaction 5: */
on receiving break()
  remove sender from the tree
  for each d_i: destTable(d_i).via = sender do
    destTable(d_i).state := broken;
  end
  if local has no child in the tree then
    send break() to parent;
  endif

/* Transaction 6: */
on receiving reject()
  remove sender from the tree
  for each d_i: destTable(d_i).via = sender do
    destTable(d_i).state := rejected;
  end
  if local has no child in the tree then
    send reject() to parent;
  endif

/* Transaction 7: */
on receiving deny(destS)
  remove sender from the tree
  for each d_i: destTable(d_i).via = sender do
    destTable(d_i).state := unknown;
  end
  set state of (local, sender) as unusable;
  execute Transaction 2.3;
```

Following are the details of the algorithm:

I. Setup Phase

1. When a node receives a request (*open* message) for opening a multicast connection with parameters S and Δ, it becomes the source node s of the multicast connection (i.e., the only tree node in the multicast tree).

   1.1 The source node then selects a next node among its neighbors for each destination in the set of destinations under consideration by the source node. Denote the set of destinations under consideration by a node v as $S'(v)$. For each destination $d_i \in S'(v)$, a table called *destTable* records the following information: *state* - the local state for d_i, and *via* - the "best" neighbor node along which there is a potentially feasible cost based shortest path to reach d_i. Initially, both fields are "*unknown*". For source node, all destinations should be included in the set of destinations under consideration. So, $S'(s) = S$. Field *via* for each destination in destTable is set to the "best" neighbor node.

1.2 For each selected neighbor (i.e., it appears in field *via* of destTable at least once), a *setup* message is sent to the neighbor in an attempt to include the neighbor into the multicast tree. This *setup* message carries a set of destinations to be covered via the link to that neighbor and the accumulated delay from the source node *s*. The *state* fields for the corresponding destinations in destTable are set to "*setup*".

2. When a node *v* receives a *setup* message, it includes itself into the tree.

2.1 If the addition of node *v* to the tree will introduce a loop (which means that node *v* is already in the tree), node *v* will do the following to avoid the loop:

2.1.1 If the accumulated delay along the existing path from the source node *s* to all destination nodes to be covered via the link to node *v* is within the delay constraint, then node *v* sends a *reject* message to the sender of the *setup* message.

2.1.2 Else if the new accumulated delay from the source node *s* to node *v* via the sender is less than the old accumulated delay, then node *v* sends a *break* message to node *v*'s parent to break the existing tree path and accept the new one.

2.1.3 Otherwise, node *v* sends a *deny* message to the sender of the *setup* message, so that the sender can re-expand the tree to cover those denied destinations.

2.2 If node *v* is a destination itself, it sends a *notify* message to the source node *s*.

2.3 Regardless, whether node *v* itself is a destination or not, if there are destinations yet to be considered (i.e., $S'(v) - \{v\} \neq \varnothing$), then node *v* sets field *via* for each destination in destTable to the "best" neighbor node, and sends a *setup* message to the selected neighbors in order to include the neighbors into the multicast tree. This

is achieved in a way that is the same as 1.1 and 1.2. Note that if there is no potentially feasible cost based shortest path that can be found for a destination, node v sends a *destination* message to the source node to indicate that the specified destination cannot be covered in Phase I.

3. When the source node s receives the *destination* message, it adds the specified destination node to the uncovered destination list *US*.

   If all destination nodes have been processed, the algorithm checks if the uncovered destination list *US* is empty. If this is the case, the algorithm terminates. Otherwise, the adjustment phase (Phase II) is invoked.

4. When the source node s receives the *notify* message, it adds the received destination node to the covered destination list *CS*.

   If all destination nodes have been processed, the algorithm checks if the uncovered destination list *US* is empty. If this is the case, the algorithm terminates. Otherwise, the adjustment phase (Phase II) is invoked.

5. When node v receives the *break* message, it removes the sender node from the multicast tree. It marks the *state* of those destinations in $S'(v)$ that are already included in the tree via the sender node as "*broken*". If the sender node is its only child, then the *break* message will be forwarded further to its parent node.

6. When node v receives the *reject* message, it removes the sender node from the multicast tree and marks the *state* of those destinations in $S'(v)$ that have been set up via the sender node as "*rejected*". If the sender node is its only child, then the *reject* message will be forwarded further to its parent node.

7. When node v receives the *deny* message, it removes the sender node from the multicast tree, resets the *state* of those destinations in $S'(v)$ that are already included in the tree via the sender node as "*unknown*" and marks the link (v, *sender*) as "*unusable*".

   It then expands the tree as in 2.3.

II. Adjustment Phase

The algorithm in Phase II is very similar to what is described in Phase I, except the following differences:

- The set of destinations under consideration in the source node is US instead of S.
- The criterion for choosing the "best" neighbor node for a destination becomes the neighbor node along which there is a feasible delay based shortest path.
- *setup* message is replaced with *adjust* message.
- On receiving *break* and *reject* message, if all destination nodes have been processed and the uncovered destination list is not empty, the algorithm terminates with failure.

Now, let us determine whether the proposed algorithm DCSP generates a constrained multicast tree. The answer is affirmative, and it is proved by the following theorem.

Theorem 4.1: When DCSP successfully constructs a multicast tree, the multicast tree is constrained; that is, the tree satisfies the delay constraint.

Proof: According to the algorithm (regardless its phase being Phase I or Phase II), a node v can expand the multicast tree to node w only if for each $d \in S'(v)$ that is going to be routed via w, $P(v) + D(v,w) + SD(w,d) < \Delta$, where $SD(w,d)$ = minimum delay from w to d; $D(v,w)$ = delay on the link (v,w); $P(v)$ = delay on path from s to v in the tree. So, every node that is included into the multicast tree must satisfy the delay constraint.

In case that w is already in the tree, as the algorithm allows w to become node v's child node only if the new accumulated delay value $P(w)$ is smaller than the older one, it is guaranteed that all destinations that have been chosen to be routed via w will continue to satisfy the delay constraint. Therefore, if the algorithm successfully builds a multicast tree, it will be a delay constrained one. EOP.

Next, let us discuss the performance of DCSP in terms of the number of message exchanges and the convergence time, and the quality of the generated multicast tree by DCSP in terms of the tree cost. The following theorem shows that in the worst case, DCSP has a lower message and time complexity than DKPP, and a lower time complexity than DSPH. The simulation conducted in the next section will complement the analysis made here by comparing DCSP with DKPP and DSPH further in terms of the number of message exchanges and the convergence time in the average case. The simulation also makes the comparison between these three algorithms in terms of the tree cost and tree construction success rate.

Theorem 4.2: DCSP's message complexity is O(mn) and time complexity is O(n) in the worst case.

Proof: During the setup phase, the *setup* message for each destination will be sent at most n times. Since there are m destinations, there will be at most O(mn) number of *setup* messages. The *notify*, *destination* and *deny* message will not be sent more than m times as only one *notify* or *destination* or *deny* message can be sent for each destination. The *break* or *reject* messages can be sent at most n times along the way to each destination. Similarly, during the adjustment phase, all types of messages can be sent no more than O(mn) times. So, the algorithm runs with a message complexity of O(mn) messages.

As the algorithm expands the tree to reach all destinations concurrently, the setup (or adjust) message will reach each reachable destination in O(n) time during the setup (or adjustment) phase. So, the algorithm runs with a time complexity of O(n). EOP.

Compared with DKPP's message complexity of O(n^3) and time complexity of O(n^3) [KPP96], DCSP has a much better message and time complexity. DSPH runs with O(mn) number of messages and time as it constructs a tree to cover each destination sequentially. So, DCSP has a better time complexity than DSPH.

To complete the discussion, we want to mention that the handling of the dynamic membership (i.e., set of destination nodes S is changing during a multicast session) in the proposed algorithm can be done by a similar approach that have been used in the existing distributed delay-constrained multicast routing algorithms such as DSPH [Jia98]. When a destination node is added, the source s can expand the tree along a delay-constrained cost based shortest path from s to the destination node. When a destination node is removed, the destination node can be removed if the node is a leaf node of the tree, or it can be marked as a non-destination node if the node is a non-leaf node of the tree.

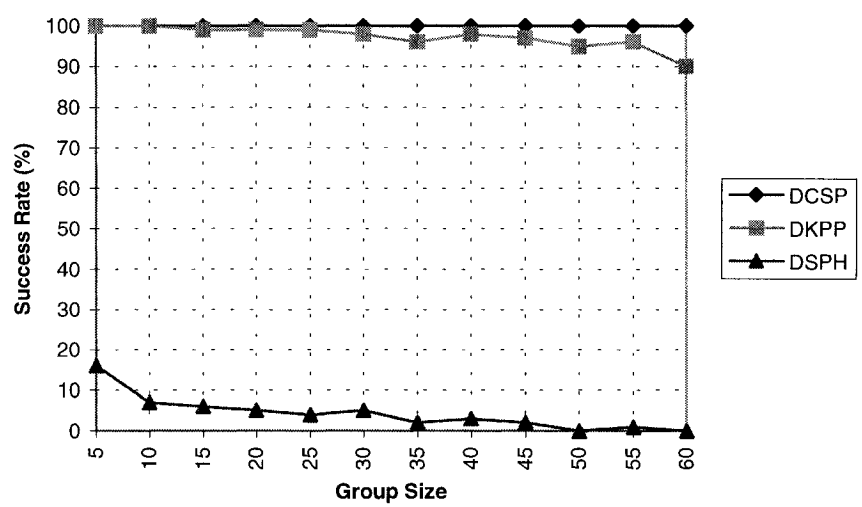
4.3 Performance analysis

In order to compare the performance of DCSP with DKPP and DSPH, a series of simulations have been performed by applying DCSP, DKPP and DSPH to networks generated by the Waxman's approach, as explained in Section 1.4.

The number of messages exchanged, the convergence time and the cost of the multicast tree are measured by their average value in a total of 100 simulation runs on a network with 200 nodes. Δ is set to $d_{max} + (i/8)d_{max}$, where $d_{max} = \max(\{d_u \mid \text{for any } u \in S$:

d_u is the delay on the delay based shortest path from s to u}) and i is an integer between 1 and 15. Note that an exchanged message will only be counted once from its sender to its receiver no matter how many intermediate nodes are walked through as long as the algorithm running on the node does not interpret the message. Meanwhile, the convergence time is counted by taking one message exchange as a time unit. However, within one time unit, there may be several message exchanges occurring in the network. Thus, multiple messages exchanged within the same time unit will be considered as one message exchange when calculating the convergence time.

Figure 4.2: Success rate versus group size when delay is fixed at i = 3



The first group of simulations are conducted by fixing the delay at i = 3 and letting the size of a multicast group change from 5 to 60 in order to see how the algorithms perform when the group size changes. Figures 4.2, 4.3, 4.4 and 4.5 show the success rate, tree cost, number of messages and time versus group size. From Figure 4.2, we can see that under the delay constraint at i = 3, DSPH has a near zero percent success rate for almost all group size. This is a serious problem for DSPH, which means that DSPH is not

useful when the delay constraints are tight. As a result, DSPH cannot be compared in the following 3 figures for tree cost, number of messages and time. For showing if it is worth optimizing the cost of multicast trees, the tree cost shown in Figure 4.3 also shows the cost of delay based shortest path trees (indicated by SPTd), which could be considered as the delay constrained multicast trees without any optimization on tree costs. It shows that the optimization significantly reduces tree costs by 20 percent.

Figure 4.3: Tree cost versus group size when delay is fixed at $i = 3$

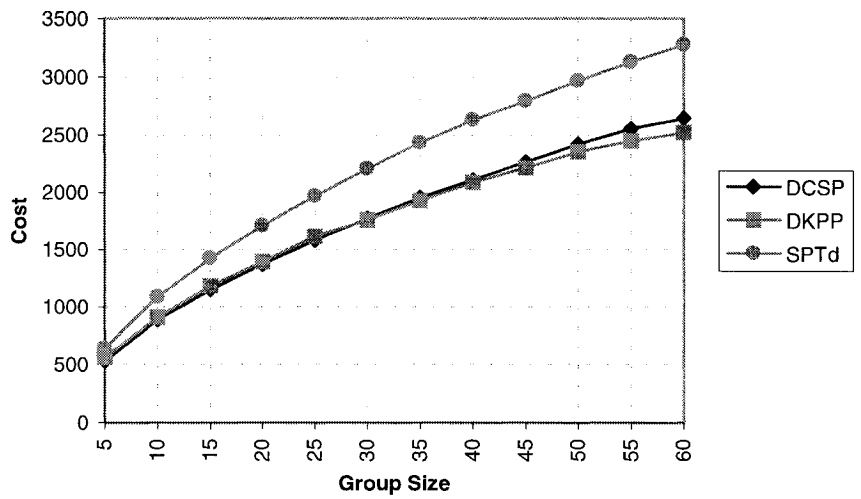
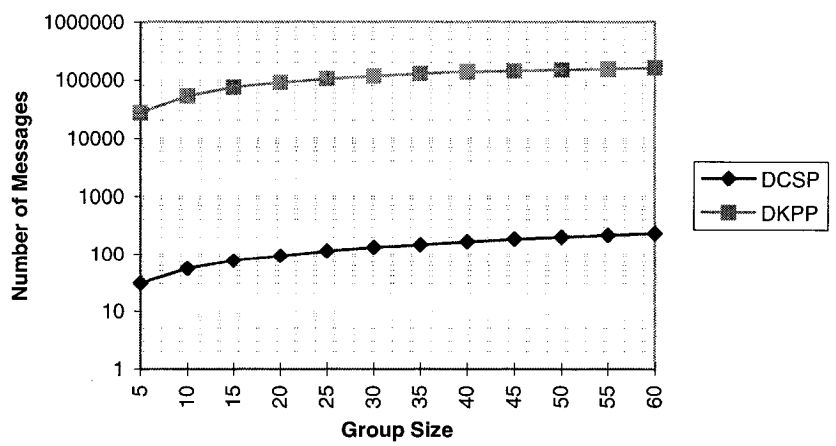
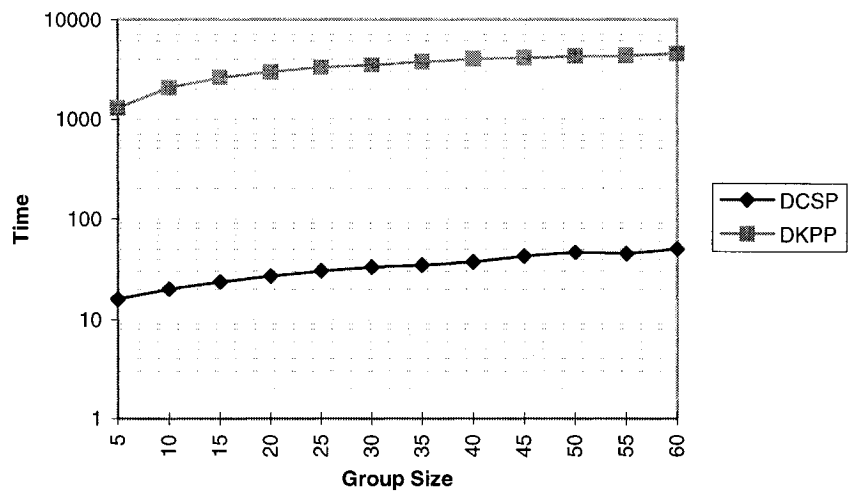


Figure 4.4: Number of messages versus group size when delay is fixed at $i = 3$



From Figure 4.3, we see that the tree cost generated by DCSP and DKPP are almost identical. However, there is a big difference between DCSP and DKPP in terms of the number of messages and time as it can be seen in Figure 4.4 and Figure 4.5, respectively. The number of messages used by DKPP algorithm is between 705 and 987 times more than that by DCSP, while the time required by DKPP is between 82 to 110 times more than that by DCSP. It also indicates that the bigger the group size, the worse the success rate of DKPP, while DCSP almost has 100% success rate for all group sizes. So, it can be concluded that the performance of DCSP is much better than that of DKPP in terms of success rate, number of messages and time.

Figure 4.5: Time versus group size when delay is fixed at i = 3



The second group of simulations are conducted by fixing the group size at 20 nodes and letting the delay change from i = 1 to 15 in order to see how the algorithms perform when the delay constraint changes. Figures 4.6, 4.7, 4.8 and 4.9 shows the success rate, tree cost, number of messages, and time versus the delay constraints,

respectively. The figures show that the cost of multicast trees generated by DCSP are as optimal as DKPP. However, the number of messages used by DKPP is between 666 and 2182 times more than that by DCSP, while the time needed by DKPP is between 68 and 399 times more than that by DCSP. Also, with respect to the success rate observed in our simulation study as shown in Figure 4.6, DKPP may fail to construct a multicast tree under tight delay constraints, while DCSP can always successfully construct a delay constrained multicast tree.

Figure 4.6: Success rate versus delay when group size = 20

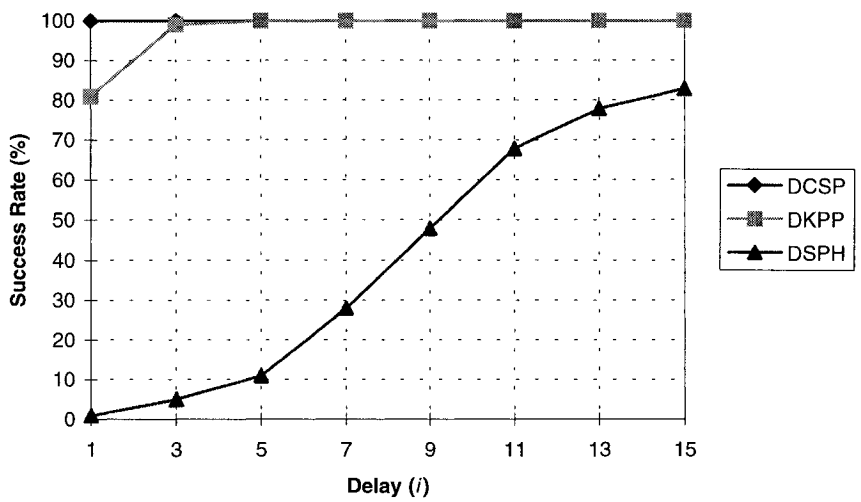
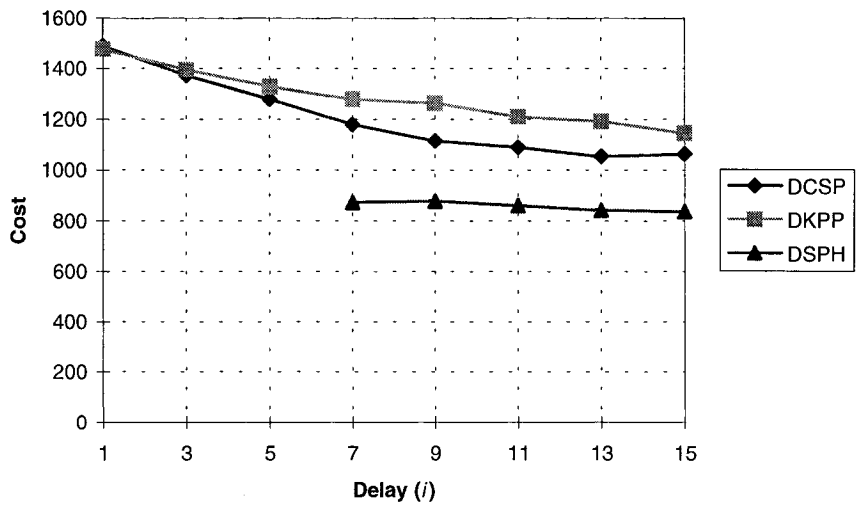


Figure 4.7: Tree cost versus delay when the group size = 20



As for the tree cost, Figure 4.7 shows that the cost of multicast trees generated by DSPH is better than both DKPP and DCSP. But, the success rate of DSPH is very low under the tight delay constraint, which undermines the optimality of its tree costs. Furthermore, the time complexity of DSPH is much higher than that of DCSP, while DCSP has similar number of messages exchanged to that of DSPH.

Figure 4.8: Number of messages versus delay when group size = 20

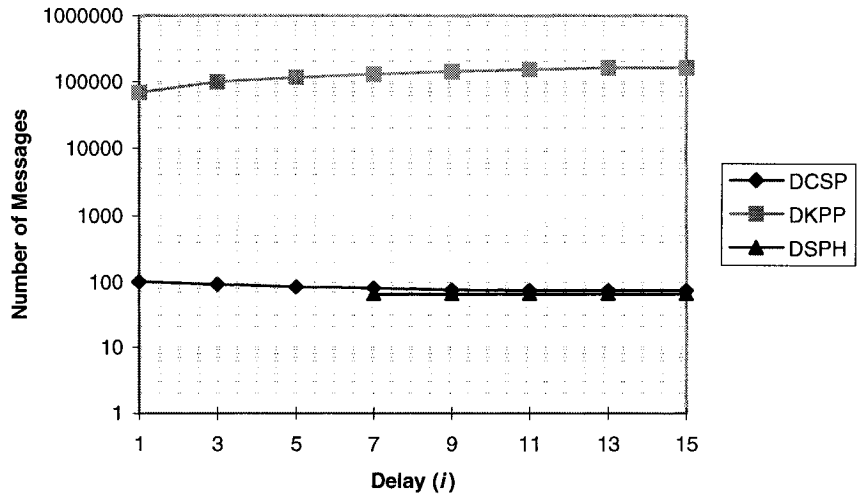
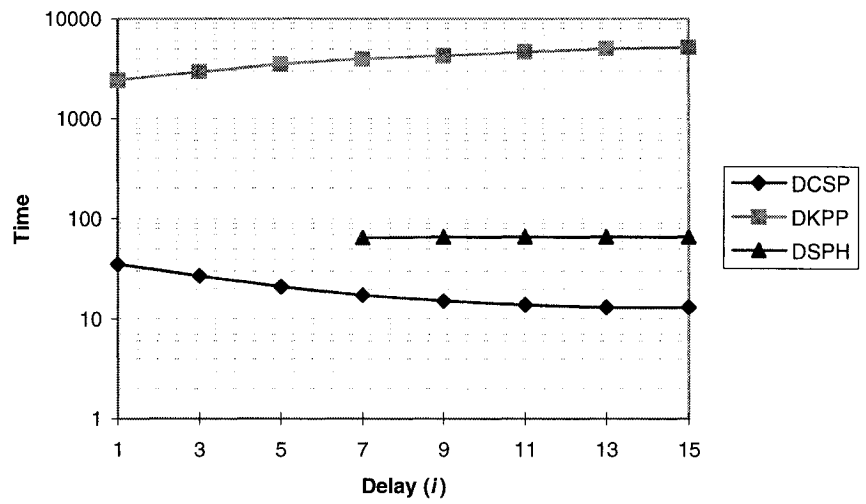


Figure 4.9: Time versus delay when group size = 20



In conclusion, compared with DKPP, DCSP has better performance in terms of number of messages, time and success rate, while produces equal quality of multicast trees in terms of the tree cost. Compared with DSPH, DCSP has the advantage in time and success rate, while performs as well in terms of the number of messages exchanged. The better tree cost by DSPH is obtained at the cost of very poor success rate.

4.4 Conclusions

In this chapter, we proposed an efficient distributed delay-constrained multicast routing algorithm which constructs a multicast tree to cover all destination nodes in a concurrent manner. It has been observed that the existing algorithms have several deficiencies. An MST based algorithm like DKPP runs with a high complexity of $O(n^3)$ in the number of messages and time. Secondly, such an algorithm has a very low success rate in constructing a constrained multicast tree especially under tight delay constraints. On the

other hand, an SP based algorithm like DSPH has a fatal deficiency; that is, the algorithm will not be able to find a solution if the delay constraint is less than the maximum delay of a cost based shortest path tree. Lastly, both algorithms try to construct the multicast tree sequentially without taking advantage of the concurrency in a distributed algorithm.

The proposed algorithm builds the multicast tree in concurrent manner which results in a very low time complexity. It also maintains low complexity in the number of messages, high success rate and high quality of multicast trees in terms of tree costs. Compared with DKPP, the proposed algorithm gives better performance in terms of the success rate, the tree cost, the number of exchanged messages and the convergence time. It is interesting to see that the high complexity of DKPP does not translate into better quality of trees (i.e., smaller tree cost). Compared with DSPH, the proposed algorithm solves the delay constrained multicast tree problem in the cases where DSPH is not able to. Also, it gives better performance in terms of convergence time and performs as well in terms of the number of message exchanges. Both advantages could be critical to some real-time applications which require the construction time for a multicast tree to be short and the delay constraint to be tight.

Chapter 5

Distributed Delay Constrained Multicast Routing Algorithm with Efficient Fault Recovery

5.1 Introduction

In the last chapter, we presented an efficient distributed delay constrained multicast routing algorithm (called as *DCSP*) which builds the multicast tree in a concurrent fashion. Algorithm DCSP has a very low message and time complexity of O(mn) and O(n) respectively, and has a near zero failure rate in constructing a multicast tree even under very tight delay constraints without any limitation on the value of delay constraints. The simulation results show that algorithm DCSP significantly outperforms the existing distributed delay constrained multicast routing algorithms which construct the multicast tree in a sequential fashion and thus do not take advantage of the concurrency in the underlying distributed computation.

In this chapter, algorithm DCSP is augmented with a fault recovery approach that takes into consideration the node failures in a network and recovers from these failures. Though we only consider the node failures in the proposed algorithm, the approach can easily be extended to cover other types of changes in the topology of the network such as link failures. The analysis and simulation results show that the augmented algorithm not only does the fault recovery effectively, but also performs the fault recovery efficiently, which gives better performance in terms of the number of exchanged messages and the convergence time than the fault recovery approach used in the existing distributed delay

constrained multicast routing algorithms [Jia98] in a network where node failures occur. There are two major benefits for having such a fault tolerant algorithm: First, it will make the fault recovery actions transparent to applications during the construction of a multicast tree and during an on-going multicast session. Second, it will re-connect the disconnected sub-tree of a multicast tree without interrupting the running traffic on the rest of the multicast tree.

The rest of this chapter is organized as follows. Section 5.2 introduces the augmentation of algorithm DCSP with an effective and efficient fault recovery approach; Section 5.3 discusses the performance of the augmented algorithm; and finally Section 5.4 gives the conclusions.

5.2 A concurrent distributed multicast routing algorithm with fault recovery

5.2.1 Overview of the proposed augmentation

Like the existing delay constrained distributed multicast routing algorithms, DCSP assumes that there are no network topology changes during the construction of a multicast tree and the on-going multicast session. If the network topology changes, DCSP will fail to complete the delay constrained multicast tree. DCSP depends on a naive fault recovery approach, which simply waits for applications to re-start the algorithm from scratch. This makes DCSP, like DKPP and DSPH, take longer time to complete when failures occur.

Our aim is to design an effective and efficient fault recovery approach which will make the recovery transparent to the applications, will not interrupt the running traffic on the rest of the multicast tree, and will shorten the time to recover from node failures. We also want the fault recovery approach to be integrated into DCSP such that the augmented

DCSP recovers from node failures by adaptively constructing a constrained multicast tree when node failures occur.

In addition, we want DCSP augmented with the fault recovery approach, called henceforth *Adaptive distributed Concurrent Shortest Path heuristic* (ACSP), to generate the delay constrained multicast trees whose quality is as good as that of DCSP in terms of tree cost, and perform as well as DCSP in terms of message complexity. These two goals are in general conflicting with each other. For example, to be able to do fault recovery, it is generally anticipated that extra messages will be needed.

The key idea that will be used in our fault recovery approach is to localize the recovery action to the failed portion of the multicast tree without re-building the whole tree again. Then, the problem is how the information about the failed sub-tree can be communicated to the nodes which are participating in the computation, so that the new information can be taken into consideration in the rest of the tree construction process. One straightforward approach is to flood the network with the information. However, this approach requires $O(n^2)$ messages just for the notification of the node failure alone. We do not take this approach. In ACSP, the failed sub-tree is refrained from flooding the network with fault information messages, and the fault information is propagated through the regular tree setup messages and follow-up messages as much as possible.

With respect to the main control steps of the algorithm, ACSP progresses in the same way as DCSP. Between the steps, ACSP checks if any tree node fails. When a failure is detected, it removes the sub-tree rooted at the failed node and notifies the source node s of the destination nodes that were covered by the removed sub-tree. Thus, the source node s is enabled to add these removed destination nodes later as ACSP will report back

to the source node s when all the remaining destination nodes have been added to the multicast tree.

Furthermore, loops may be introduced in the multicast tree due to the network topology changes. ACSP will detect these loops by checking if a node (say v) to be added is already in the tree. A loop is removed by choosing the path from the source node s to node v which has the minimum delay. This will ensure that all existing paths between the source node s and destinations that go through node v still satisfy the delay constraint. If the new parent of v has the minimum delay from the source node, node v will accept the new parent and break itself from its previous parent. Otherwise, node v will reject the new parent.

After the tree is constructed, ACSP will continue to run the tree node failure checking and take recovery steps to repair the delay constrained multicast tree if node failures occur during an ongoing multicast session.

Informally, given a network $G=(V, E)$, a source node s, a set of destination nodes S, and a delay constraint Δ, ACSP progresses in one or two phases as follows (additional steps related to our fault recovery approach are given in bold):

Phase I - Setup

The same as in Phase I of DCSP except the additional step that is inserted between step 4 and 5.

1. Starts with a tree containing only the source node s;
2. Selects one of its neighbors as a "best" neighbor for each destination node to be covered, and expands the tree concurrently along these best neighbors. The "best"

neighbor for a destination node will be the neighbor along which there is a potentially feasible cost based shortest path that can reach the destination;

3. If there is no potentially feasible cost based shortest path found, adds the destination node under consideration to the list *N* of destinations which are not yet covered by the tree;
4. If a node is included into the tree by more than one parent node, the node will choose the parent node which has the smallest accumulated delay from the source node;
5. **Checks if any tree node fails. When a failure is detected, removes the sub-tree rooted at the failed node and returns the destination nodes covered by the sub-tree back to the list of destinations to be covered by the tree;**
6. Repeats steps 2-5 until each node in *S* is either included in the tree or included in the list *N* of destinations which are not yet covered by the tree.

Phase II - Adjustment

The same as in Phase II of DCSP except the additional step that is inserted between step 4 and 5.

1. Starts from the source node with a tree containing all nodes *S* - *N* that are included in Phase I;
2. Selects a "best" neighbor for each destination node to be covered, and expands the tree concurrently along these best neighbors. The "best" neighbor for a destination node in this stage will be the neighbor along which there is a feasible delay based shortest path;
3. If there is no feasible delay based shortest path found, adds the destination node under consideration to the list *M* of destinations which are not yet covered by the tree;

4. If a node is included into the tree by more than one parent node, the node will choose the parent node which has the smallest accumulated delay from the source node;
5. **Checks if any tree node fails. When a failure is detected, removes the sub-tree rooted at the failed node and returns the destination nodes covered by the sub-tree back to the list of destinations to be covered by the tree;**
6. Repeats steps 2-5 until each node in N is included in the tree or included in the list M of destinations which are not yet covered by the tree.

5.2.2 Details of the augmentation

Besides the seven types of messages in DCSP, a new type of message is used in ACSP, which is

remove - removing a sub-tree from the tree.

It is assumed that node failures are detected by a lower level protocol. Following are the details of the augmentation (in bold):

I. Setup Phase

1. When a node receives a request (*open* message) for opening a multicast connection with parameters S and Δ, it becomes the source node s of the multicast connection (i.e., the only tree node in the multicast tree).
   1.1 The source node then selects a next node among its neighbors for each destination in the set of destinations under consideration by the source node. Denote the set of destinations under consideration by a node v as $S'(v)$. For each destination $d_i \in S'(v)$, a table called *destTable* records the following information: *state* - the local state for d_i, and *via* - the "best" neighbor node along which there is a potentially feasible cost based shortest path to reach d_i. Initially, both fields are "*unknown*".

For source node, all destinations should be included in the set of destinations under consideration. So, $S'(s) = S$. Field *via* for each destination in destTable is set to the "best" neighbor node.

1.2 For each selected neighbor (i.e., it appears in field *via* of destTable at least once), a *setup* message is sent to the neighbor in an attempt to include the neighbor into the multicast tree. This *setup* message carries a set of destinations to be covered via the link to that neighbor and the accumulated delay from the source node *s*. The *state* fields for the corresponding destinations in destTable are set to "*setup*".

2. When a node *v* receives a *setup* message, it includes itself into the tree.

2.1 If the addition of node *v* to the tree will introduce a loop (which means that node *v* is already in the tree), node *v* will do the following to avoid the loop:

2.1.1 If the accumulated delay along the existing path from the source node *s* to all destination nodes to be covered via the link to node *v* is within the delay constraint, then node *v* sends a *reject* message to the sender of the *setup* message.

2.1.2 Else if the new accumulated delay from the source node *s* to node *v* via the sender is less than the old accumulated delay, then node *v* sends a *break* message to node *v*'s parent to break the existing tree path and accept the new one.

2.1.3 Otherwise, node *v* sends a *deny* message to the sender of the *setup* message, so that the sender can re-expand the tree to cover those denied destinations.

2.2 If node *v* is a destination itself, it sends a *notify* message to the source node *s*.

2.3 Regardless, whether node v itself is a destination or not, if there are destinations yet to be considered (i.e., $S'(v) - \{v\} \neq \varnothing$), then node v sets field *via* for each destination in destTable to the "best" neighbor node, and sends a *setup* message to the selected neighbors in order to include the neighbors into the multicast tree. This is achieved in a way that is the same as 1.1 and 1.2. Note that if there is no potentially feasible cost based shortest path that can be found for a destination, node v sends a *destination* message to the source node to indicate that the specified destination cannot be covered in Phase I.

3. When the source node s receives the *destination* message, **it adds the specified destination node to the uncovered destination list *US* if it is because no feasible path can be found or *FS* if it is because a node failure occurred.**

   If all destination nodes have been processed, **the algorithm checks if the uncovered destination list *US* and *FS* is empty.** If this is the case, the algorithm terminates. **If *FS* is non-empty, it expands the tree as in 1.** Otherwise, the adjustment phase (Phase II) is invoked.

4. When the source node s receives the *notify* message, it adds the received destination node to the covered destination list *CS*.

   If all destination nodes have been processed, **the algorithm checks if the uncovered destination list *US* and *FS* is empty.** If this is the case, the algorithm terminates. **If *FS* is non-empty, it expands the tree as in 1.** Otherwise, the adjustment phase (Phase II) is invoked.

5. When node v receives the *break* message, it removes the sender node from the multicast tree. It marks the *state* of those destinations in $S'(v)$ that are already included

in the tree via the sender node as "*broken*". If the sender node is its only child, then the *break* message will be forwarded further to its parent node.

6. When node v receives the *reject* message, it removes the sender node from the multicast tree and marks the *state* of those destinations in $S'(v)$ that have been set up via the sender node as "*rejected*". If the sender node is its only child, then the *reject* message will be forwarded further to its parent node.
7. When node v receives the *deny* message, it removes the sender node from the multicast tree, resets the *state* of those destinations in $S'(v)$ that are already included in the tree via the sender node as "*unknown*" and marks the link (v, *sender*) as "*unusable*".

   It then expands the tree as in 2.3.
8. **When node v in the tree detects that its child node fails, node v removes the child node from the tree.**
9. **When node v in the tree detects that its parent node fails or receives the *remove* message, node v removes itself from the tree. If node v is not a leaf node, it will forward the *remove* message to all of its children. If node v is a destination node, it sends a *destination* message to the source node s so that the source node s will add the destination node back to the uncovered destination list FS.**

II. Adjustment Phase

The algorithm in Phase II is very similar to what is described in Phase I, except the following differences:

- The set of destinations under consideration in the source node is US instead of S.
- The criterion for choosing the "best" neighbor node for a destination becomes the neighbor node along which there is a feasible delay based shortest path.

- *setup* message is replaced with *adjust* message.
- On receiving *break* and *reject* message, if all destination nodes have been processed and the uncovered destination list is not empty, the algorithm terminates with failure.

The correctness of the proposed algorithm is shown by the following theorem.

Theorem 5.1: The delay-constrained multicast tree built by ACSP is loop free.

Proof: If there is a loop in the tree, there must be a node which has two parent nodes. In ACSP, each time a node receives a *setup* message for joining the tree, the node checks if it is already in the tree. If it is not in the tree yet, it will join the tree and set the sender of the *setup* message as its parent node. Otherwise, it will either reject the *setup* message or accept the *setup* message but break from its existing parent node first. So, in all cases, each node has one and only one parent node. Therefore, the delay-constrained multicast tree built by ACSP is loop free. EOP.

Now, let us determine whether the proposed algorithm generates a constrained multicast tree. The answer is affirmative, and it is proved by the following theorem.

Theorem 5.2: When ACSP terminates with a multicast tree, the multicast tree is constrained; that is, the tree satisfies the delay constraint.

Proof: According to the algorithm, a node v can expand the multicast tree to node w only if for the destination $d \in S$ that is going to be routed via w, $P(v) + D(v,w) + SD(w,d) < \Delta$, where $SD(w,d)$ = minimum delay of the shortest path from w to d; $D(v,w)$ = delay on the link (v,w); $P(v)$ = delay on path from s to v in the tree. So, every node that is included into the multicast tree must satisfy the delay constraint.

In case that w is already in the tree, as the algorithm allows w to become node v's child node only if the new accumulated delay value $P(w)$ is smaller than the older one, it is

guaranteed that all destinations that have been chosen to be routed via w will continue to satisfy the delay constraint. Therefore, if the algorithm successfully builds a multicast tree, it will be a delay constrained one. EOP.

Next, let us discuss the performance of ACSP in terms of the number of message exchanges and the convergence time, and the quality of the generated multicast tree by ACSP in terms of the tree cost. Without node failures, DCSP runs with O(mn) message and O(n) time complexity in the worst case. As each node failure will make DCSP re-run, DCSP will have a O(kmn) message and O(kn) time complexity if there are k-1 node failures during a multicast session. The following theorem shows that in the worst case, ACSP performs as good as DSPH in terms of the message and time complexity. The simulation conducted and reported in the next section will complement the analysis made here by comparing ACSP with DCSP further in terms of the number of message exchanges and the convergence time in the average case. The simulation also makes the comparison between these two algorithms in terms of the tree cost.

Theorem 5.3: ACSP's message complexity is O(kmn) and time complexity is O(kn) in the worst case, where k-1 is the number of node failures occurred in the network during the construction of a delay constrained multicast tree and the on-going multicast session.

Proof: The k-1 node failures force ASPH to try k times to complete the construction of a delay constrained multicast tree. Each time, the *setup* message for each destination will be sent O(n) times. Since there are m destinations, there will be O(mn) number of *setup* messages. The *destination* message will not be sent more than m times as only one *destination* message can be sent for each destination. The *break* or *reject* messages can be sent O(n) times along the way to each destination. The *remove* messages are sent once

along each edge of the sub-tree rooted at the failed node, which is O(n) times. So, the algorithm runs with a message complexity of O(kmn).

The setup message will reach all destinations in O(n) time for each trial of completing a delay constrained multicast tree. So, the algorithm runs with a time complexity of O(kn). EOP.

5.3 Performance analysis

In order to compare the performance of ACSP with DCSP, a series of simulations have been performed by applying DCSP and ACSP to networks generated by the Waxman's approach as explained in Section 1.4.

The number of messages exchanged, the convergence time and the cost of the multicast tree are measured by their average value in a total of 100 simulation runs on a network with 200 nodes. Δ is set to $d_{max} + (i/8)d_{max}$, where $d_{max} = \max(\{d_u \mid \text{for any } u \in S$: d_u is the delay on the delay based shortest path from s to $u\}$) and i is an integer between 1 and 15. Note that an exchanged message will only be counted once from its sender to its receiver no matter how many intermediate nodes are walked through as long as the algorithm running on the node does not interpret the message. Meanwhile, the convergence time is counted by taking one message exchange as a time unit. However, within one time unit, there may be several message exchanges occurring in the network. Thus, multiple messages exchanged within the same time unit will be considered as one message exchange when calculating the convergence time.

We compare the performance of ACSP with DCSP in the average case under the condition that a node failure occurs during the construction of a delay constrained

multicast tree or during the on-going multicast session. DCSP is going to use the naïve fault recovery approach; that is, DCSP will be re-run from scratch when a node failure occurs.

In the simulations, node failures are injected into networks in order to see how the multicast routing algorithms perform in a network where node failures occur. There are two types of experiments: one is for the case when node failures occur during the construction of a multicast tree, and the other is for the case when node failures occur during the on-going multicast session. For the first type of experiments, it is assumed that at most one node failure may occur during the construction of the multicast tree. The timing for a node failure is randomly selected so it could occur randomly among the different stages of the construction of the multicast tree. The failed node is randomly selected among the nodes in the multicast tree built so far that are neither the source node nor the destination nodes when node failure occurs, since the failure of the source node means that there will be no multicast tree to be built and the failure of a destination node means that the constructed multicast tree will not be comparable with other multicast trees which cover all destination nodes. For the second type of experiments, DCSP will have to be re-run to re-build the entire multicast tree when a node failure occurs during an ongoing multicast session. In the experiments, one multicast tree node will be randomly selected as a failure node during a multicast session for the type of networks that we study. As DCSP will be re-run when a node failure occurs while ACSP will always return a multicast tree no matter whether a node failure occurs or not, ACSP has a better success rate than DCSP.

Figure 5.1, Figure 5.2 and Figure 5.3 show the simulation results when Δ is set to $d_{max} + (3/8)d_{max}$, where $d_{max} = \max(\{d_u \mid \text{for any } u \in S: d_u \text{ is the delay on the shortest path from } s \text{ to } u\})$, and the group size changes between 5 and 60 in 200-node networks. In the figures, suffix "-c" means during the construction of a multicast tree, suffix "-m" means during the on-going multicast session and "SPT-d" means the delay-based SPT. The delay-based SPT could be considered as the delay constrained multicast trees without any optimization on tree costs.

Figure 5.1: Tree cost versus group size when node failure occurs

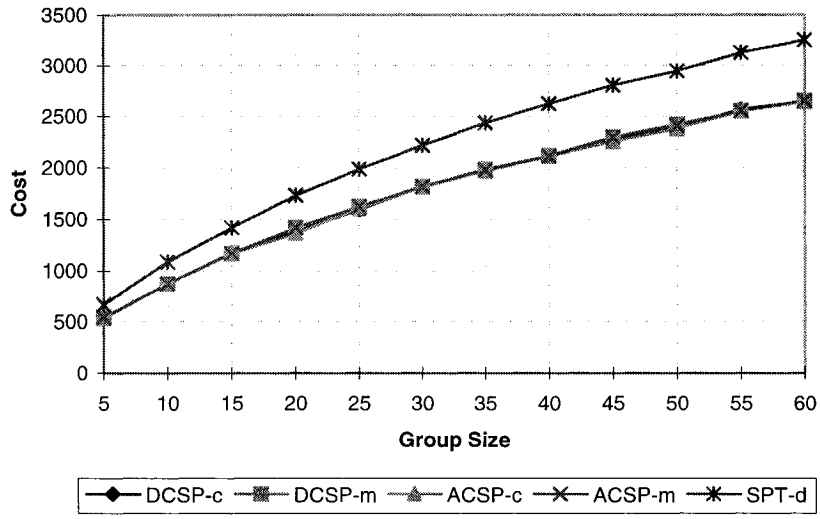


Figure 5.1 shows that the costs of the trees generated by ACSP are almost identical to those by DCSP. This result is very encouraging. DCSP calculates the multicast tree based on the consistent current network topology information after it re-runs while ACSP might use different network topology information for different parts of a delay constrained multicast tree. Intuitively, it could be expected that the costs of the trees generated by ACSP should be noticeably higher than those by DCSP. But, the simulation results show that it is not the case. It is also shown in Figure 5.1 that the delay-constrained

multicast tree algorithms generate trees with much better cost performance than the algorithms without considering optimization on the tree cost such as SPT-d. This means that it is worth using the delay-constrained multicast tree algorithms rather than using SPT-d directly.

Figure 5.2: Number of messages versus group size when node failure occurs

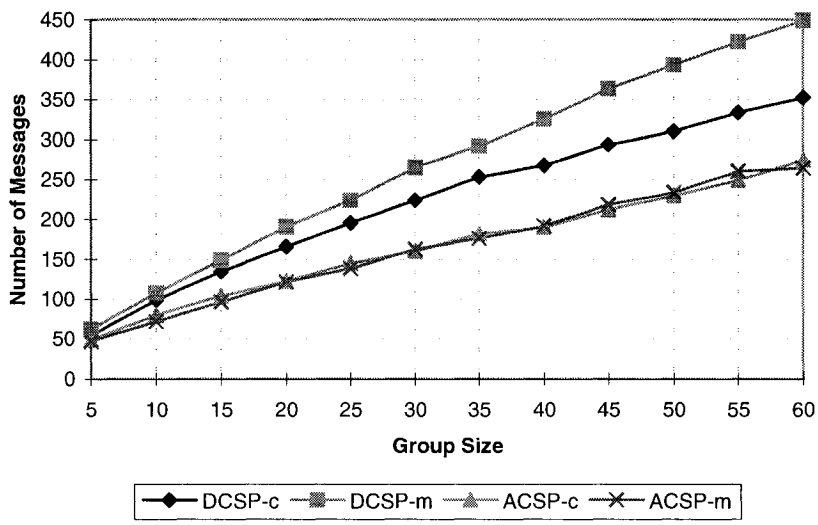


Figure 5.3: Time versus group size when node failure occurs

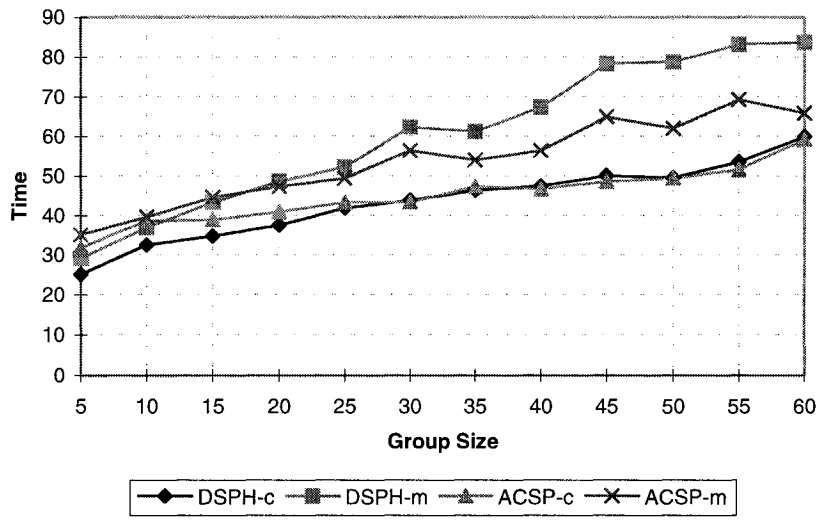


Figure 5.2 shows that the number of messages required by DCSP is up to 28% more than that required by ACSP during the construction of a delay constrained multicast

tree, and is up to 70% more than that required by ACPH during the on-going multicast session. This result is surprising as we know that doing fault recovery normally costs extra number of messages. Intuitively, one would expect that because DCSP is very efficient on using messages, any fault recovery approach that tries to merge the list of uncovered destinations in the failed sub-tree with the list of uncovered destinations in the participating node will have a higher number of messages than DCSP even though DCSP has to run twice. Contrary to this expectation, ACSP has a significantly better performance than DCSP in terms of the number of messages. This is due to the fact that ACSP uses the approach that refrains from sending messages on node failure. The previous analysis shows that the delay on sending node failure messages does not have a negative effect on the quality of the generated multicast trees.

Figure 5.3 shows that the convergence time required by DCSP is up to 27% more than that by ACSP during the on-going multicast session. This result confirms what has been expected. Through the localized recovery approach taken by ACSP, it is expected that the convergence time can be reduced by ACSP.

5.4 Conclusions

In this chapter, algorithm DCSP is augmented with a fault recovery approach that takes into account the changes in the topology of the network. The augmented algorithm can recover from node failures during the construction of a delay constrained multicast tree, and during an on-going multicast session without requiring rebuilding of the entire multicast tree. Furthermore, compared with the distributed delay constrained multicast routing algorithm that uses the naïve fault recovery approach, the augmented algorithm

gives better performance in terms of the number of exchanged messages and the convergence time when applied in a network where node failures occur.

multicast tree will be "expand the tree from the local node along the cost based shortest path to a destination node if the delay constraint is satisfied; otherwise expand the tree along the delay based shortest path to the destination node". This process can be completed in O(n) messages and time. It is different from DSPH's criterion which is "expand the tree along the cost based shortest path from the tree to a destination node" that requires the source node to query each node in the tree to decide the best tree node for tree expansion, and costs O(mn) messages and O(n) time. Also, the criterion that we will use allows the proposed algorithm to include a cost based shortest path to a destination as much as possible in order to make the tree cost as low as possible. It is important to note that conflicts may occur if a node on the path towards the new destination node is invited more than once to join the tree. The solution is that the node where the conflicts occur will choose its parent whose accumulated delay from the source node is minimal to ensure that all destinations that go through this node will satisfy the delay constraint.

If a sub-tree of new destination nodes wants to join the multicast tree, the root node of the sub-tree will send a join message to the source node of the tree. The join message will carry the maximum delay time in the sub-tree (say Δ') requesting to join the tree. The source node then initiates the process to expand the tree to cover the new sub-tree of destination nodes in the same way as it tries to cover a single new destination node, except that the delay constraint becomes $\Delta - \Delta'$ instead of Δ.

If an existing destination node wants to leave the multicast tree, it sends a leave message towards the source node along the tree and the nodes on the path will be removed until a fork node or another destination node is reached.

We call this dynamic multicast routing algorithm as *Distributed Dynamic Concurrent Shortest path heuristic* (DDCS) since it handles the dynamic group membership. The dynamic multicast routing algorithm in the existing distributed delay constrained multicast routing algorithm DSPH (which is the one discussed in Chapter 2) is referred as *Distributed Dynamic Shortest Path heuristic* (DDSP). DDSP will be used for comparison as it has the best performance among all SP-based distributed dynamic multicast routing algorithms which we have seen.

6.2.2 Details of the proposed algorithm

Besides the seven types of messages in DCSP, two new types of messages are used in DDCS, which are

join - notifying the source node for joining the tree;

leave - notifying its parent node for leaving the tree.

Following are the details of the algorithm:

LEAVE

1. When a node v receives a request (*leave event*) for leaving the multicast tree, it checks if it has any child node. If there is not a child node, it sends a *leave* message to its parent node and removes itself from the tree. Otherwise, the algorithm terminates.
2. When a node v receives a *leave* message, it checks if the sender is its child node, it has only one child node and it is not a destination node. If all conditions are true, it removes the sender from its child list, sends a *leave* message to its parent node and removes itself from the tree. Otherwise, the algorithm terminates.

JOIN

1. When a node *v* receives a request (*join event*) for joining the multicast tree, it sends a *join* message to the source node of a multicast tree.
2. When source node *s* receives a *join* message, it adds the sender to *S* and set it as the only node in the set of destinations under consideration by the source node.
   2.1 The source node then selects a next node among its neighbors for each destination in the set of destinations under consideration by the source node. Field *via* for each destination in destTable is set to the "best" neighbor node.
   2.2 For each selected neighbor (i.e., it appears in field *via* of destTable at least once), a *setup* message is sent to the neighbor in an attempt to include the neighbor into the multicast tree. This *setup* message carries a set of destinations to be covered via the link to that neighbor and the accumulated delay from the source node *s*. The *state* fields for the corresponding destinations in destTable are set to "*setup*".
3. It then expands the tree starting at Step 2 of Algorithm DCSP 's Phase I (given in Chapter 4).

As it was done in the case of DCSP, it is straightforward to show that the delay-constrained multicast tree built by DDCS is loop free, and the proposed algorithm DDCS generates a constrained multicast tree.

Next, let us discuss the performance of DDCS in terms of the number of message exchanges and the convergence time, and the quality of the generated multicast tree by DDCS in terms of the tree cost. For the deletion of an existing destination node, DDSP runs with O(n) message and time complexity in the worst case, while for the addition of a new destination node, DDSP runs with O(mn) message and O(n) time complexity in the

worst case. The following theorem shows that in the worst case, DDCS performs better than DDSP in terms of the message and time complexity. The simulation conducted and reported in the next section will complement the analysis made here by comparing DDCS with DDSP further in terms of the number of message exchanges and the convergence time in the average case. The simulation also makes the comparison between these two algorithms in terms of the tree cost.

Theorem 6.1: For the deletion of an existing destination node and the addition of a new destination node, DDCS's message and time complexity is O(n) in the worst case.

Proof: When deleting an existing destination node from a tree, the *leave* message will be sent at most O(n) times since a path from any tree node to the source node (i.e., the root of the tree) cannot contain more than n links. So, the message and time complexity for DDCS to deleting an existing destination node is O(n) in the worst case.

As we know that DCSP runs with O(mn) message and O(n) time complexity in the worst case, where m is the number of destination nodes. The addition of a new destination node in DDCS is a special case of DCSP when m = 1. So, the message and time complexity for DDCS to adding a new destination node is O(n) in the worst case. EOP.

6.3 Performance analysis

In order to compare the performance of DDCS with DDSP , a series of simulations have been performed by applying DDCS and DDSP to networks generated by the Waxman's approach, as explained in Section 1.4.

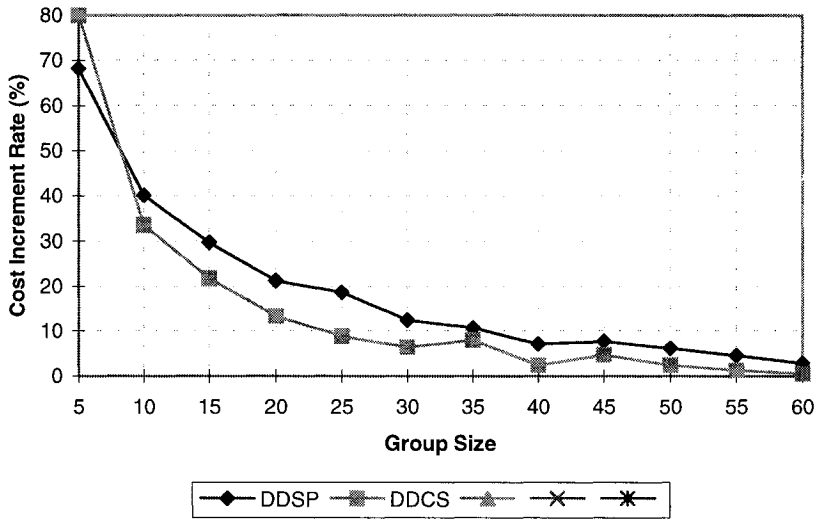
The number of messages exchanged, the convergence time and the cost of the multicast tree are measured by their average value in a total of 100 simulation runs on a

network with 200 nodes. Δ is set to $d_{max} + (i/8)d_{max}$, where $d_{max} = \max(\{d_u \mid \text{for any } u \in S$: d_u is the delay on the cost based shortest path from s to $u\}$) and i is an integer between 1 and 15. Note that an exchanged message will only be counted once from its sender to its receiver no matter how many intermediate nodes are walked through as long as the algorithm running on the node does not interpret the message. Meanwhile, the convergence time is counted by taking one message exchange as a time unit. However, within one time unit, there may be several message exchanges occurring in the network. Thus, multiple messages exchanged within the same time unit will be considered as one message exchange when calculating the convergence time.

We compare the performance of DDCS with DDSP in the average case when the initial multicast group size changes. For each initial group size, 100 simulations are run. For each simulation run, a series of 10 addition (join) or deletion (leave) requests occur after the initial multicast tree construction. The probability of an addition request is related to N_t, the number of nodes in the tree, by the function $\gamma(200- N_t)/[\ \gamma(200 - N_t) + (1 - \gamma)\ N_t]$. The value of γ determines the equilibrium point at which the probability of an addition or deletion is equally likely. In our simulations, γ is set to 0.3. So, approximately the same number of addition and deletion requests are generated in the simulation run when the maximum initial group size 60 is approached.

Figure 6.1, Figure 6.2 and Figure 6.3 show the simulation results when Δ is set to $d_{max} + (3/8)d_{max}$, where $d_{max} = \max(\{d_u \mid \text{for any } u \in S$: d_u is the delay on the shortest path from s to $u\}$), and the initial group size changes between 5 and 60 in 200-node networks. In the figures, suffix "-a" means during the addition of a new destination node and suffix "-d" means during the deletion of an existing destination node.

Figure 6.1: Tree cost increment rate versus group size when group is dynamic



For group dynamic multicast routing problem, the quality of the generated multicast trees can be measured by *cost increment rate*, which is calculated by the formula $(C_a - C_0)/C_0$, where C_a is the tree cost after receiving addition/deletion requests and C_0 is the cost of the initial multicast tree. The lower the cost increment rate, the better the delay-constrained dynamic multicast routing algorithm. Figure 6.1 shows that the cost increment rates by DDCS are close to those by DDSP, and even better than those by DDSP. This could be due to that DDCS is more likely to choose an existing tree path for adding a new destination node into the tree.

Figure 6.2 shows that the number of messages required by DDSP is 5 to 20 times as many as that required by DDCS during the addition of a new destination node, and is identical to that required by DDCS during the deletion of an existing destination node. This result confirms what has been expected. Through avoiding querying the best distance in DDCS, it is expected that the number of messages can be reduced by DDCS.

Figure 6.3 shows that the convergence time required by DDSP is 2 to 3 times as many as that by DDCS during the addition of a new destination node, and is identical to that required by DDCS during the deletion of an existing destination node. This result confirms what has been expected. Through avoiding querying the best distance in DDCS, it is expected that the convergence time can be reduced by DDCS.

Figure 6.2: Number of messages versus group size when group is dynamic

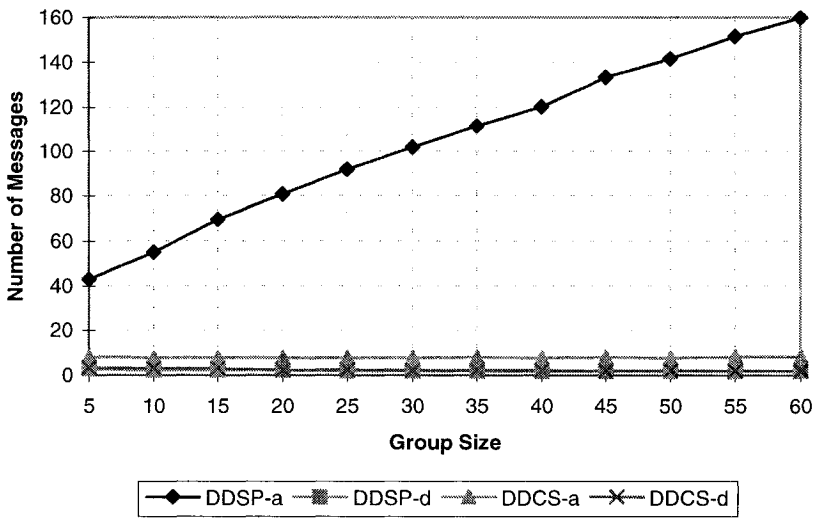
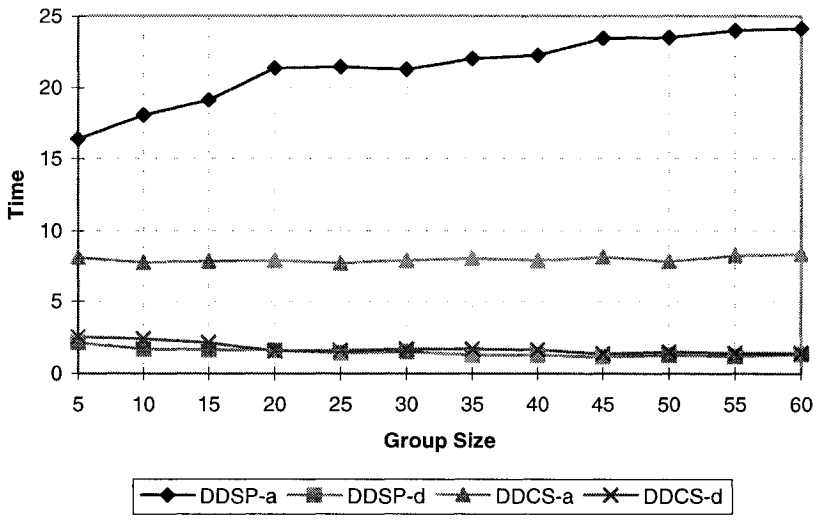


Figure 6.3: Time versus group size when group is dynamic



6.4 Conclusions

In this chapter, algorithm DCSP is expanded to a delay constrained dynamic multicast routing algorithm DDCS. The proposed algorithm can handle the addition or deletion of a multicast destination node without requiring rebuilding of the entire multicast tree. Furthermore, compared with the existing distributed delay constrained dynamic multicast routing algorithm, the proposed algorithm gives better performance in terms of the number of exchanged messages and the convergence time when applied in a network where the multicast group changes dynamically.

Chapter 7

Conclusions and Final Remarks

In this thesis, we have identified and studied several important issues for solving the QoS-based (specifically delay constrained) multicast routing problem. In most cases, the multicast routing environment is dynamic. The dynamic multicast routing environment means that the network topology changes [Ram00]. For example, a new node is added, an existing node or link fails, or the cost values associated with links change. There are many research results for the topology change problem in the unicast routing but few have been reported for the multicast routing problem (i.e., topology dynamic multicast routing problem). The goal in this research is to find a distributed algorithm for the topology dynamic multicast routing problem which should be efficient in terms of the message and time complexity, should generate high quality multicast trees in terms of tree cost, and finally the generated multicast trees should satisfy QoS requirements such as delay constraints.

Secondly, the dynamic multicast routing problem also means that the multicast group can be dynamic, that is, new destination nodes (i.e., group members) may be added to the group, and existing destination nodes may be removed from the multicast group. For example, in a live audio or video news conference, users may frequently join in or leave. We can call this problem as the *group dynamic multicast routing problem* to distinguish it from the topology dynamic multicast routing problem. The challenge is to

construct a sequence of minimum cost trees given a sequence of destination node addition and removal requests. The research in this area is still limited. Our goal is to find a distributed algorithm for the group dynamic multicast routing problem which should be efficient in terms of the message and time complexity, should generate a sequence of high quality multicast trees in terms of tree cost when the multicast group changes, and finally each generated multicast tree should satisfy QoS requirements such as delay constraints.

Lastly, we try to address the most basic problem that is to find a more efficient distributed delay constrained multicast routing algorithm in terms of the message and time complexity even when the network topology and the multicast group are static. It has been observed that the existing delay constrained multicast routing algorithms try to construct a multicast tree by following the approach of adding destination nodes one by one, which has a very low efficiency. The solution for this basic problem in turn helps us to solve the topology and group dynamic multicast routing problem efficiently.

Our contributions for solving the delay constrained multicast routing problem start with the following two delay constrained multicast routing algorithms considering the dynamic network topology changes:

- a new distributed delay constrained multicast routing algorithm that is SP-based and capable of doing a local fault recovery from node or link failures in the network. The performance evaluation shows that the proposed algorithm performs better in terms of message and time complexity than an existing SP-based distributed delay constrained multicast routing algorithm which has to conduct a complete fault recovery from node failures in the network.

- a new distributed delay constrained multicast routing algorithm that is MST-based and capable of fault recovery from node failures in the network. The performance evaluation shows that the proposed algorithm performs better in terms of message and time complexity than an existing MST-based distributed delay constrained multicast routing algorithm which has to conduct a complete fault recovery from node failures in the network.

The approach used in ASPH for recovering from link failures can be adapted to the other algorithms presented in the thesis in a straightforward manner. Hence, the adaptation of the approach to the remaining algorithms are not given in the thesis.

The next important contribution is a new distributed delay constrained multicast routing algorithm that is more efficient than any existing distributed delay constrained multicast routing algorithms in terms of time complexity. The efficiency of the proposed algorithm is achieved by covering all destination nodes concurrently instead of covering each destination node sequentially as in the traditional distributed delay constrained multicast routing algorithms.

The further contributions are made by proposing the following two distributed delay constrained multicast routing algorithms considering the dynamic network topology and group membership changes:

- a new distributed delay constrained multicast routing algorithm that augments our proposed concurrent distributed delay constrained multicast routing algorithm and capable of doing a local fault recovery from node failures in the network. The performance evaluation shows that the proposed algorithm performs better in terms of message and time complexity than the original concurrent distributed delay constrained

multicast routing algorithm which has to conduct a complete fault recovery from node failures in the network.

- a new distributed delay constrained dynamic multicast routing algorithm that expands our proposed concurrent distributed delay constrained multicast routing algorithm and capable of handling the addition and deletion of destination nodes in the multicast group. The performance evaluation shows that the proposed algorithm performs better in terms of message and time complexity than the existing distributed delay constrained dynamic multicast routing algorithm.

There are some open issues that require further research. All QoS-based multicast routing algorithms studied here assume that the precise network state information is available for calculating the multicast tree. However, network state information is inherently imprecise in a distributed network environment, which affects the routing performance [CN98]. For large networks, network nodes are clustered into groups forming a hierarchical network structure to make the network scalable. Only the aggregated network state information is available to the nodes in other groups. Therefore, it is required to further investigate efficient distributed QoS-based multicast routing algorithms that will take the hierarchical structure of the network topology into consideration and also take the imprecise network state information into consideration.

REFERENCES

[Bal97] A. Ballardie, "Core based trees (CBT version 2) multicast routing protocol specification," RFC 2189, Internet Engineering Task Force, September 1997.

[BCZ98] A. Ballardie, B. Cain and Z. Zhang, "Core based trees (CBT version 3) multicast routing protocol specification," Internet Draft draft-ietf-idmr-cbt-spec-v3-01, Internet Engineering Task Force, August 1998.

[BFC93] A. J. Ballardie, P. Francis, and J. Crowcroft, "Core Based Trees (CBT): An architecture for scalable inter-domain multicast routing," Proceedings of ACM SIGCOMM'93, 1993.

[BG92] D. Bertsekas and R. Gallager, *Data Networks*, Second Edition, Englewood Cliffs: Prentice-Hall, 1992.

[Bil97] T. Billhartz, J. B. Cain, E. Farrey-Goudreau, D. Fieg, and S. G. Batsell, "Performance and resource cost comparisons for the CBT and PIM multicast routing protocols," IEEE Journal on Selected Areas in Communications, vol. 15, no.3, pp.304-315, April 1997.

[BJ83] K. Bharath-Kumar and J. M. Jaffe, "Routing to multiple destinations in computer networks," IEEE Transactions on Communications, vol. 31, no. 3, March 1983.

[BV95] F. Bauer and A. Varma, "Degree-constrained multicasting in point-to-point networks," in Proceedings of IEEE INFOCOM'95, pp.369-376, April 1995.

[BV96] F. Bauer and A. Varma, "Distributed algorithms for multicast path setup in data networks," IEEE/ACM Transactions on Networking, vol.4, no.2, pp.181-191, April 1996.

[BV97] F. Bauer and A. Varma, "ARIES: A rearrangeable inexpensive edge-based on-line Steiner algorithm," IEEE Journal on Selected Areas in Communications, vol. 15, no. 3, pp.382-397, April 1997.

[CC97] K. Carlberg and J. Crowcroft, "Building shared trees using a one-to-many joining mechanism," ACM SIGCOMM Computer Communication Review, vol. 27, no. 1, pp. 5-11, January 1997.

[CDZ97] K. L. Calvert, M. B. Doar and E. W. Zegura, "Modeling Internet Topology," IEEE Communications Magazine, vol. 35, no. 6, pp. 160-163, June 1997.

[CLR92] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, Cambridge, Ma: MIT, 1992.

[CNS00] S. Chen, K. Nahrstedt, and Y. Shavitt, "A QoS-aware multicast routing protocol," IEEE Journal on Selected Areas in Communications, vol. 18, no. 12, pp. 2580-2592, December 2000.

[CN98] S. Chen and K. Nahrstedt, "An overview of quality of service routing for next-generation high-speed networks: Problems and solutions," IEEE Network, pp. 64-79, November/December 1998.

[DDC97] C. Diot, W. Dabbous, and J. Crowcroft, "Multipoint communication: A survey of protocols, functions, and mechanisms," IEEE Journal on Selected Areas in Communications, vol. 15, no. 3, pp. 277-290, April 1997

[Dee88] S. Deering, "Multicast routing in internetworks and extended LANs," ACM Computer Communications Review, vol. 18, no. 4, pp. 55-64, 1988.

[Dee96] S. Deering *et al.*, "The PIM architecture for wide-area multicasting," IEEE/ACM Transactions on Networking, vol. 4, no. 2, pp. 153-162, April 1996.

[Dee97] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, A. Helmy, L. Wei, "Protocol Independent Multicast Version 2, Dense Mode Specification," Internet Draft, draft-ietf-idmr-pim-dm-06.txt, August 1997.

[Dij59] E. W. Dijkstra, "A note on two problems in connection with graphs," Numerische Mathematik, vol. 1, pp. 269-271, 1959.

[DL93] M. Doar and I. Leslie, "How bad is naive multicast routing," Proceedings of IEEE INFOCOM'93, pp. 82-89, 1993.

[DM78] Y. K. Dalal and R. M. Metcalf, "Reverse path forwarding of broadcast packets," Communications of the ACM, vol. 21, no. 12, pp. 1040-1048, December 1978.

[Doa96] M. B. Doar, "A better model for generating test networks," Proceedings of GLOBECOM'96, pp. 86-93, 1996.

[FBP98] M. Faloutsos, A. Banerjea, and R. Pankaj, "QoSMIC: Quality of service sensitive multicast Internet protocol," Proceedings of ACM SIGCOMM'98, pp. 144-153, September 1998.

[FG00] A. Fei and M. Gerla, "Receiver-initiated multicasting with multiple QoS constraints," Proceedings of INFOCOM'2000, pp. 62-70, 2000.

[Flo62] R. W. Floyd, "Algorithm 97: Shortest paths," Communications of the ACM, vol. 5, p.345, 1962.

[GHS83] R.G. Gallager, P.A. Humblett, and P.M. Spira, "A distributed algorithm for minimum-weight spanning trees," *ACM Transactions on Programming Languages and Systems,* vol. 5, pp. 66-77, 1983.

[HC99] H. W. Holbrook and D. R. Cheriton, "IP multicast channels: EXPRESS support for large-scale single-source applications," Proceedings of ACM SIGCOMM'99, 1999.

[HLP98] S.-P. Hong, H. Lee and B. H. Park, "An efficient routing algorithm for delay-sensitive applications with dynamic membership," Proceedings of IEEE INFOCOM'98, pp. 1433-1440, 1998.

[HR92] F. K. Hwang and D. Richards, "Steiner Tree Problems," Networks, vol. 22, pp.55-89, 1992.

[IW91] M. Imase and B. Waxman, "Dynamic Steiner tree problems," SIAM Journal on Discrete Mathematics, vol. 4, no. 3, pp. 369-384, August 1991.

[Jia98] X. Jia, "A distributed algorithm of delay-bounded multicast routing for multimedia applications in wide area networks," IEEE/ACM Transactions on Networking, vol. 6, pp. 828-837, Dec. 1998.

[Kar72] R.M. Karp, "Reducibility among combinatorial problems," in *Complexity Computer Communications*, R.E. Miller and J.W. Thatcher (editors), pp.85-103, New York: Plennum, 1972.

[KMB81] L. Kou, G. Markowsky, and L. Berman, "A fast algorithm for Steiner trees," Acta Informatica, no.15, pp.141-145, 1981.

[KPP92] V.P. Kompella, J.C. Pasquale, and G.C. Polyzos, "Multicasting for multimedia applications," IEEE INFOCOM'92, pp.2078-2085, 1992.

[KPP93] V.P. Kompella, J.C. Pasquale, and G.C. Polyzos, "Multicast routing for multimedia communication," IEEE/ACM Transactions on Networking, vol.1, no.3, pp.286-292, June 1993.

[KPP93a] V.P. Kompella, J.C. Pasquale, and G.C. Polyzos, "Two distributed algorithms for the constrained Steiner tree problem," in Proceedings of the Second International Conference on Computer Communications and Networking, pp.343-349, 1993.

[KPP96] V.P. Kompella, J.C. Pasquale, and G.C. Polyzos, "Optimal multicast routing with quality of service constraints," Journal of Network and Systems Management, pp.107-131, June 1996.

[Kru56] J. Kruskal, "On the shortest spanning subtree of a graph and the traveling salesman problems," Proceedings of the American Mathematical Society, vol. 7, pp. 48-50, 1956.

[Kum98] S. Kumar *et al.*, "The MASC/BGMP architecture for inter-domain multicast routing," Proceedings of ACM SIGCOMM'98, 1998.

[Law76] E. Lawler, *Combinatorial Optimization: Networks and Matroids*, New York: Holt, Rinehart and Winston, 1976.

[LS02] C.P. Low and X. Song, "On finding feasible solutions for delay constrained group multicast routing problem," IEEE Transactions on Computers, vol.51, no.5, pp.581-588, May 2002.

[Max97] N. F. Maxemchuk, "Video Distribution on Multicast Networks," IEEE Journal on Selected Areas in Communications, vol. 15, no. 3, pp. 357-372, 1997.

[Moy94] J. Moy, "Multicast routing extensions for OSPF," Communications of the ACM, vol. 37, no. 8, pp. 61-66, August 1994.

[NST99] P. Narvaez, K.-Y. Siu and H.-Y. Tzeng, "New dynamic SPT algorithm based on a ball-and-string model," Proceedings of IEEE INFOCOM'99, 1999.

[NT94] C. A. Noronha and F. A. Tobagi, "Evaluation of multicast routing algorithms for multimedia streams," in Proceedings of IEEE International Telecommunication Symposium, August 1994.

[Pau98] S. Paul, *Multicasting on the Internet and its applications*, Kluwer Academic Publishers, 1998.

[Perl99] R. Perlman, C-Y Lee, A. Ballardie, J. Crowcroft, Z. Wang, T. Maufer, C. Diot, J. Thoo and M. Green, "Simple Multicast: a Design for Simple, Low-Overhead Multicast," Internet Draft, draft-perlman-simple-multicast-03.txt, October 1999.

[PG97] M. Parsa and J. J. Garcia-Luna-Aceves, "A protocol for scalable loop-free multicast routing," IEEE Journal on Selected Areas in Communications, vol. 15, no. 3, pp. 316-331, April 1997.

[Prim57] R. Prim, "Shortest connection networks and some generalizations," *Bell Systems Tech. J.*, vol.36, pp.1389-1401, 1957.

[Ram00] M. Ramalho, "Intra- and inter-domain multicast routing protocols: a survey and taxonomy," IEEE communications Surveys & Tutorials, Electronic magazine at http://www.comsoc.org/pubs/surveys, vol. 3, no. 1, 2000.

[RB97] G. N. Rouskas and I. Baldine, "Multicast routing with end-to-end delay and delay variation constraints," IEEE Journal on Selected Areas in Communications, vol. 15, no. 3, pp. 346-356, April 1997.

[RC86] V. J. Rayward-Smith and A. Clare, "On finding Steiner vertices," Networks, vol. 16, pp. 283-294, 1986.

[RS83] V. J. Rayward-Smith, "The computation of nearly minimal Steiner trees in graphs," International Journal of Mathematical Education in Science and Technology, vol. 14, no. 1, pp. 15-23, 1983.

[SC00] L. Schwiebert and R. Chintalapati, "Improved fault recovery for core based trees," Computer Communications 23, pp. 816-824, 2000.

[SMM99] R. Sriram, G. Manimaran and C. S. R. Murthy, "A rearrangeable algorithm for the construction of delay-constrained dynamic multicast trees," Proceedings of IEEE INFOCOM'99, 1999.

[SRV97] H. F. Salama, D. S. Reeves, and 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. 332-345, April 1997.

[Sun95] Q. Sun and H. Langendoerfer, "Efficient multicast routing algorithm for delay-sensitive applications," in Proceedings of 2nd international Workshop on Protocols for Multimedia Systems (PROMS'95), pp.452-458, 1995.

[Sun98] Q. Sun and H. Langendoerfer, "An efficient delay-constrained multicast routing algorithm," Journal of High-Speed Networks, vol.7, no.1, pp.43-55, 1998.

[Sun99] Q. Sun and H. Langendoerfer, "A distributed delay-constrained dynamic multicast routing algorithm," Telecommunication Systems, vol.11, pp.47-58, 1999.

[TM80] H. Takahashi and A. Matsuyama, "An approximate solution for the Steiner problem in graphs," Mathematica Japonica, vol. 24, no. 6, pp. 573-577, 1980.

[UZ01] H. Ural and K. Zhu, "Fault Recovery for a Distributed QoS-based Multicast Routing Algorithm," International Conference on Computer Networks and Mobile Computing (ICCNMC 2001) , pp.396-403, 2001.

[UZ02a] H. Ural and K. Zhu, "Fault Recovery for a distributed SP-based delay constrained multicast routing algorithm," 16th International Parallel & Distributed Processing Symposium (IPDPS 2002) , pp.452-459, Ft. Lauderdale, Florida, USA, April 2002.

[UZ02b] H. Ural and K. Zhu, "An Efficient Distributed QoS-based Multicast Routing Algorithm," 21st IEEE International Performance, Computing, and Communications Conference (IPCCC 2002) , pp.27-36, Phonix, Arizona, USA, April 2002.

[Vos92] S. Voβ, "Steiner's problem in graphs: heuristic methods," Discrete Applied Mathematics, vol. 40, pp. 45-72, 1992.

[VPL98] S. Verma, R. K. Pankaj and A. Leon-Garcia, "QoS based multicast routing algorithms for real-time applications," Performance Evaluation, vol. 34, no. 4, pp. 273-294, 1998.

[Wax88] B. M. Waxman, "Routing of multipoint connections," IEEE Journal on Selected Areas in Communications, vol. 6, no. 9, pp. 1617-1622, December 1988.

[Wax93] B. M. Waxman, "Performance evaluation of multipoint routing algorithms," Proceedings of IEEE INFOCOM'93, pp. 980-986, 1993.

[WI88] B. M. Waxman and M. Imase, "Worst-case performance of Rayward-Smith's Steiner tree heuristics," Information Processing Letters, vol. 29, pp. 283-287, 1988.

[WC96] Z. Wang and J. Crowcroft, "Quality-of-service routing for supporting multimedia applications," IEEE Journal on Selected Areas in Communications, vol. 14, no. 7, pp. 1228-1234, September 1996.

[WE94] L. Wei and D. Estrin, "The trade-offs of multicast trees and algorithms," in *Proceedings of 1994 International Conference on Computer Communications and Networks*, San Francisco, CA, September 1994.

[WH00] B. Wang and J. C. Hou, "Multicast routing and its QoS extension: problems, algorithms, and protocols," IEEE Network, pp. 22-36, January/February 2000.

[Wid94] R. Widyono, "The design and evaluation of routing algorithms for real-time channels," Technical Report TR-94-024, Tenet Group, University of California at Berkeley, 1994.

[Win87] P. Winter, "Steiner problem in networks: A survey," Networks, vol. 17, pp. 129-167, 1987.

[WS92] P. Winter and M. Smith, "Path-distance heuristics for the Steiner problem in undirected networks," Algorithmica, vol. 7, no. 2-3, pp. 309-327, 1992.

[ZKC01] B. Zhang, M. M. Krunz, and C. Chen, "A fast delay-constrained multicast routing algorithm," in Proceedings of IEEE International Conference on Communications 2001 (ICC 2001), vol.9, pp. 2676 -2680, 2001.

[ZPG95] Q. Zhu, M. Parsa, and J. J. Garcia-Luna-Aceves, "A source-based algorithm for delay-constrained minimum-cost multicasting," in Proceedings of IEEE INFOCOM'95, pp.377-385, 1995.

[ZPG98] M. Parsa, Q. Zhu and 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.